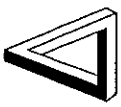
FUNDP
NAMUR

Faculté des Sciences
Département de Mathématique

Rempart de la Vierge, 8
B - 5000 Namur (Belgique)

Etude comparative d'une méthode de détermination du nombre de classes pour des données quantitatives : la Gap Statistic



Mémoire présenté pour l'obtention
du grade de
Licencié en Sciences Mathématiques
par

COSME **Sébastien**

Promoteur : HARDY André

Année Académique 2006-2007

En premier lieu, je tiens absolument à adresser mes remerciements les plus chaleureux à mon promoteur, Monsieur André Hardy, pour tous les conseils qu'il m'a prodigués, pour toutes les lumières qu'il m'a apportées, durant la conception de ce mémoire. Monsieur Hardy a pu me guider au mieux cette année, notamment en m'accordant de très intéressantes discussions à propos des points les plus délicats du sujet abordé.

Je tiens également à remercier mes proches, et plus particulièrement mes parents, mon frère Thibaut, et Pauline, pour m'avoir soutenu durant ces quatre années d'études. Leurs patience et encouragements m'ont été des plus bénéfiques.

Enfin, je tiens à exprimer ma reconnaissance envers tous les professeurs et assistants ayant jalonné ma licence, pour m'avoir transmis leurs connaissances et enthousiasme à l'égard de leurs domaines de prédilection.

Résumé

Le but de ce mémoire est d'étudier une méthode récente de détermination du nombre de classes pour des données quantitatives classiques : la **méthode de la Gap Statistic.** Il s'agira de justifier théoriquement cette méthode, puis de l'étendre aux jeux de données caractérisés par des variables symboliques dites "intervalles".
L'idée de la méthode est de comparer les variations des inerties intra-classes du jeu de données original à celles d'un jeu de données généré selon une distribution dite "nulle".
Ce mémoire contient de nombreux exemples. Sur chacune des applications, le résultat de la méthode en question est évalué ; elle est ainsi comparée à différentes autres méthodes bien connues dans la littérature.

Mots-clés : Gap Statistic, détermination du nombre de classes, variables "intervalles", inertie intra-classe, distribution nulle.

Abstract

The goal of this report is the study of a recent method to determine the number of clusters for a set of quantitative data : the **Gap Statistic method.**
This method will be theoretically justified, and then it will be extended to cope with data characterized by symbolic features, called "intervals".
The idea is to compare the changes in within cluster dispersion to these expected under an appropriate reference null distribution.
This report contains a lot of examples. On each of the applications, the output of the method is analysed ; that method is compared with other well-known methods available in the scientific literature.

Keywords : Gap Statistic, determination of the number of clusters, "interval" features, within cluster dispersion, null reference distribution.

Table des matières

Introduction

Beaucoup de méthodes visant à déterminer le nombre de classes propre à un jeu de données ont déjà été élaborées. Elles sont tantôt basées sur des critères, tantôt sur des tests. Ce mémoire a pour but d'étudier une méthode récemment développée. Il consistera à rappeler les moyens dont nous disposons aujourd'hui en matière de détermination du nombre de classes, à expliquer les étapes constituant l'algorithme de la nouvelle méthode en question, à évaluer sa qualité par l'analyse comparative de ses résultats sur différents jeux de données. Une extension au cas symbolique (particularisée aux variables intervalles) sera réalisée et sera également qualitativement étudiée.

Cette méthode est due aux auteurs R. Tibshirani, G. Walther et T. Hastie, statisticiens à l'Université de Stanford, Californie. Ce que contiennent les pages qui suivent est basé sur leur article [25]. Le fait que l'article date de l'aube de ce troisième millénaire est la preuve que la détermination du nombre de classes dans un jeu de données classique reste un problème encore fort contemporain, malgré les nombreux remarquables travaux déjà réalisés à ce propos.

De tout temps, l'homme a voulu distinguer les choses, les objets qui remplissent ou non son quotidien et ces classifications utilisées à bon escient s'avèrent naturellement être des vecteurs dirigés vers des progrès importants, quelle que soit la discipline dans laquelle ces études sont réalisées. La classification de tous les jours est par conséquent un problème propre à toute société. Cette répartition en sous-groupes d'un échantillon, voire d'une population, nécessite les réponses à deux questions cruciales.

- Comment distinguer les objets et les sous-groupes d'objets, pour les classifier de manière optimale ?

- Dans un jeu de données, combien de sous-groupes devons-nous distinguer ?

Pour ces deux questions imbriquées l'une dans l'autre, nous disposons aujourd'hui de plusieurs réponses. Il va de soi que certaines sont bien meilleures que d'autres, selon la structure des données à laquelle nous sommes confrontés. Cet ouvrage s'adresse essentiellement à la seconde question.

La qualité de telles méthodes (que ce soit pour le partitionnement du jeu de données, ou pour la détermination du bon nombre de classes) est mesurée par leur capacité à retrouver une partition intuitive en un nombre de classes intuitif. Ce raisonnement est facilement applicable aux jeux de données théoriques, conçus ou simulés de manière à ce que la structure des données soit claire, palpable. Notons que cette caractéristique est d'autant plus

claire lorsque le jeu de données est généré selon deux, voire trois variables descriptives (continues), afin qu'il soit visualisable dans un espace de représentation à deux, ou trois dimensions. Cependant, un tel confort n'est pas toujours disponible dans la collecte de données issues de la réalité. Un exemple, grand classique dans la littérature traitant de la classification, est le problème des iris de R. A. Fisher [2], évoqué ultérieurement. Ces méthodes se doivent donc d'être invariables par rapport au nombre de variables descriptives.

En ce qui concerne l'extension de la méthode au cas symbolique, rien ne permet la vérification théorique des deux méthodes construites dans ce mémoire. Nous verrons dans le chapitre en question qu'il s'agit là de premières approches uniquement basées sur des idées similaires à celles abordées lors de l'implémentation de la méthode dans le cas de données classiques.

Chapitre 1

Le problème de classification

En tout premier lieu, rappelons en quoi consiste le **problème de classification.**

Soient :

- un ensemble de n **objets** : $E = \{x_1, \ldots, x_n\}$,
- p **variables descriptives** $Y_1, \ldots, Y_p$, qui caractérisent chacun de ces objets,
- à chacune de ces variables Y_j $(j \in \{1, \ldots, p\})$ est associé un **domaine d'observation**, un ensemble image $\mathcal{Y}_j$.

Définition 1.0.1. *Une variable descriptive (ou descripteur) Y_j sur E est telle que*

$$Y_j \ : \ E \longrightarrow \mathcal{Y}_j, \qquad x_i \rightsquigarrow Y_j(x_i) = x_{ij},$$

où x_{ij} est la valeur observée de la variable Y_j sur l'objet x_i.

Le **but de la classification automatique** est de trouver k **classes naturelles** dans E, et d'attribuer une caractérisation, une **étiquette**, à chacune de ces classes.

Définition 1.0.2. *$P = \{C_1, \ldots, C_k\} \subset \mathcal{P}(E)$ est une partition de E (en k classes) si et seulement si*

1. $\forall\, l \in \{1, \ldots, k\}, \qquad C_l \neq \emptyset,$
2. $\forall\, l, m \in \{1, \ldots, k\} \ : \ l \neq m, \qquad C_l \cap C_m = \emptyset,$
3. $\sum_{l=1}^{k} C_l = E$, *où $\sum$ représente ici l'union disjointe.*

Précisément, il faudra rechercher une partition $P = \{C_1, \ldots, C_k\}$ de E (en k classes). Pour ce faire, nous choisirons un **critère de classification** qui mesure la qualité de toutes les partitions de E en k classes :

$$W \ : \ \mathcal{P}_k \longrightarrow \mathbb{R}, \qquad P \rightsquigarrow W(P, k),$$

où $\mathcal{P}_k$ est l'ensemble de toutes les partitions de E en k classes.
Deux de ces critères seront évoqués dans le chapitre suivant.

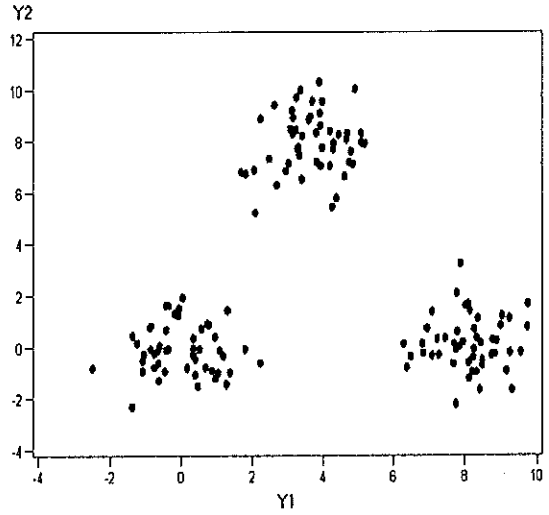


FIG. 1.1 – Trois classes intuitivement distinctes avec $n = 150$ et $p = 2$

La partition retenue sera la **partition optimale** $P^* = \{C_1^*, \ldots, C_k^*\}$, telle que

$$W(P^*, k) = \min_{P \in \mathcal{P}_k} W(P, k).$$

Avant de pouvoir aborder le reste de ce premier chapitre, il est nécessaire de définir des **distances entre objets** et des **distances entre classes** (i. e. entre groupes d'objets).

La distance séparant deux objets de E, x_u et x_v, sera la distance euclidienne d. Elle sera utilisée tout au long de ce document, sous la forme suivante :

$$d(x_u, x_v) = \sqrt{\sum_{j=1}^{p} (x_{uj} - x_{vj})^2}.$$

Notons qu'un individu x_i décrit par p variables (quantitatives continues dans le cas qui nous occupe) peut ainsi être assimilé à un point de l'espace vectoriel $\mathbb{R}^p$: $E \subset \mathbb{R}^p$.

Définissons, pour un élément C_l d'une partition P de l'ensemble E (en k classes), deux concepts primordiaux pour la suite.

Définition 1.0.3. *Soit le groupe de points C_l, élément d'une partition de l'ensemble E (en k classes). Le centroïde (ou centre de gravité) de cette classe C_l, noté $g^{(l)}$, est défini par*

$$g^{(l)} = (g_1^{(l)}, \ldots, g_p^{(l)}) \in \mathbb{R}^p, \quad \text{où} \quad g_j^{(l)} = \frac{1}{|C_l|} \sum_{x_i \in C_l} x_{ij}.$$

Définition 1.0.4. *Soit le groupe de points C_l, élément d'une partition de l'ensemble E (en k classes). L'inertie de cette classe C_l, notée $I(C_l)$, est définie par*

$$I(C_l) = \sum_{x_i \in C_l} d^2(x_i, g^{(l)}).$$

Soient maintenant C_u et C_v deux classes, éléments d'une partition de E (en k classes). Nous disposons d'une série de **mesures de dissimilarité** correspondant à ces deux classes, toutes associées à des méthodes dites ascendantes hiérarchiques, dont les algorithmes sont détaillés ultérieurement dans ce chapitre. En voici cinq exemples :

1. l'indice d'agrégation du **lien minimum** [10] :

$$\delta_1(C_u, C_v) := \min_{x \in C_u, y \in C_v} d(x, y),$$

2. l'indice d'agrégation du **lien moyen** [10] :

$$\delta_2(C_u, C_v) := \frac{1}{|C_u|.|C_v|} \sum_{x \in C_u, y \in C_v} d(x, y),$$

3. l'indice d'agrégation du **lien maximum** [10] :

$$\delta_3(C_u, C_v) := \max_{x \in C_u, y \in C_v} d(x, y),$$

4. l'indice d'agrégation des **centroïdes** [10] :

$$\delta_4(C_u, C_v) := d(g^{(u)}, g^{(v)}),$$

5. l'indice d'agrégation de **Ward** [8] :

$$\begin{aligned} \delta_5(C_u, C_v) &:= I(C_u \cup C_v) - I(C_u) - I(C_v) \\ &= \textstyle\sum_{x_i \in C_u \cup C_v} d^2(x_i, g^{(u \cup v)}) - \sum_{x_i \in C_u} d^2(x_i, g^{(u)}) - \sum_{x_i \in C_v} d^2(x_i, g^{(v)}) \\ &= \frac{|C_u|.|C_v|}{|C_u|+|C_v|} \; d^2(g^{(u)}, g^{(v)}), \qquad [18] \end{aligned}$$

où $g^{(u \cup v)} \in \mathbb{R}^p$ est le centroïde de la classe $C_u \cup C_v$.

C'est grâce à ces distances entre objets ou entre groupes d'objets qu'il nous est possible de créer des **matrices de dissimilarités**, nécessaires au bon déroulement d'un algorithme ascendant hiérarchique. Notons qu'un objet peut être considéré comme une classe formée d'un seul objet, un **singleton**. Il est par conséquent possible de calculer une dissimilarité, une "distance" entre un point et une classe.

Si le nombre total de partitions de n objets en k classes est noté par $S(n, k)$, alors

$$S(n, k) = \frac{1}{k!} \sum_{i=1}^{k} C_k^i (-1)^{k-i} i^n. \qquad [1]$$

Cette valeur croît de manière astronomique lorsque n et p augmentent. C'est pourquoi des **algorithmes de classification**, qui évitent de considérer les partitions par énumération complète, ont été developpés. Tout au long de ce mémoire, nous ne travaillerons qu'avec deux méthodes, aux principes très différents, décrites ci-après.

1.1 Classification ascendante hiérarchique (C. A. H.)

Définition 1.1.1. *Soient*
- *E l'ensemble d'objets à partitionner,*
- *H un ensemble de parties non vides de E, appelées paliers.*

H sera une hiérarchie sur E si et seulement si

1. $E \in H$ *(palier le plus haut),*
2. $\forall\, x_i \in E,\ \{x_i\} \in H$ *(singletons formant les paliers terminaux),*
3. $\forall\, h, h' \in H\ :\ h \cap h' \neq \emptyset,\quad h' \subset h \quad ou \quad h \subset h'.$

Cette hiérarchie sera dite binaire si chaque palier non terminal est formé de la réunion d'exactement deux éléments disjoints déjà dans H, i. e.

$$\forall\, h \in H : |h| > 1,\ \exists!\ (h', h'') \in H \times H \quad :\quad h' \cap h'' = \emptyset \ \ et\ \ h' \cup h'' = h.$$

La C. A. H. est une famille d'algorithmes largement utilisés en classification automatique, dont le **but** est la construction de **partitions emboitées**, de moins en moins fines, dont les classes forment une **hiérarchie binaire** [10]. L'algorithme prend en premier lieu compte des n singletons, puis les agglomère en paliers selon la distance entre classes δ utilisée, pour enfin arriver au regroupement des deux paliers qui formeront l'ensemble entier E.

Une autre famille d'algorithmes agit de manière similaire, à la différence près qu'ils considèrent en premier lieu l'ensemble E pour aboutir en fin de procédure à une partition en n classes, celle des singletons. C'est la **classification descendante hiérarchique** (C. D. H.), non abordée ici.

Voici l'énoncé d'un algorithme typiquement C. A. H., basé sur un **indice d'agrégation** δ, semblable aux cinq indices vus précédemment dans ce premier chapitre :

Algorithme général de la C. A. H.

1. Initialisation
   $P^{(0)} := \{C_1^{(0)}, \ldots, C_n^{(0)}\}$ où $C_i^{(0)} = \{x_i\}$, avec $\delta : \mathcal{P}(E) \times \mathcal{P}(E) \longrightarrow \mathbb{R}^+$ comme *indice d'agrégation*, dissimilarité entre groupes d'objets.

2. Etape agrégative
   A la $k^{\text{ème}}$ étape, construire $P^{(k)}$ contenant $n-k$ classes, à partir de $P^{(k-1)}$ contenant $n-(k-1) = n-k+1$ classes en réunissant les deux classes les plus proches au sens de δ. Mettre ensuite à jour les $\frac{(n-k)(n-k+1)}{2}$ distances restant à considérer.
   Recommencer jusqu'à l'agrégation des deux classes formant l'ensemble E, i. e. $P^{(n-1)}$.

Remarque

Si, à l'étape d'agrégation, plus d'un couple de groupes de points minimisent simultanément l'indice δ, le premier d'entre eux rencontré par l'algorithme sera sélectionné. La binarité de la hiérarchie en output sera par conséquent assurée.

Il est possible de développer autant d'algorithmes C. A. H. qu'il existe d'indices d'agrégation, dont cinq ont été cités précédemment dans ce chapitre. Appliqués à l'algorithme établi dans cette section, ils donnent lieu aux méthodes du lien minimum, du lien moyen, du lien maximum, des centroïdes, de Ward. Ces méthodes sont applicables à tous les jeux de données, mais les partitions obtenues peuvent différer de la partition naturelle. Nous le verrons dans le quatrième chapitre.

Voyons maintenant les outputs de ces cinq méthodes appliquées au jeu de données hypersphérique basique exposé en début de chapitre. Ces outputs sont obtenus à partir du logiciel SAS.

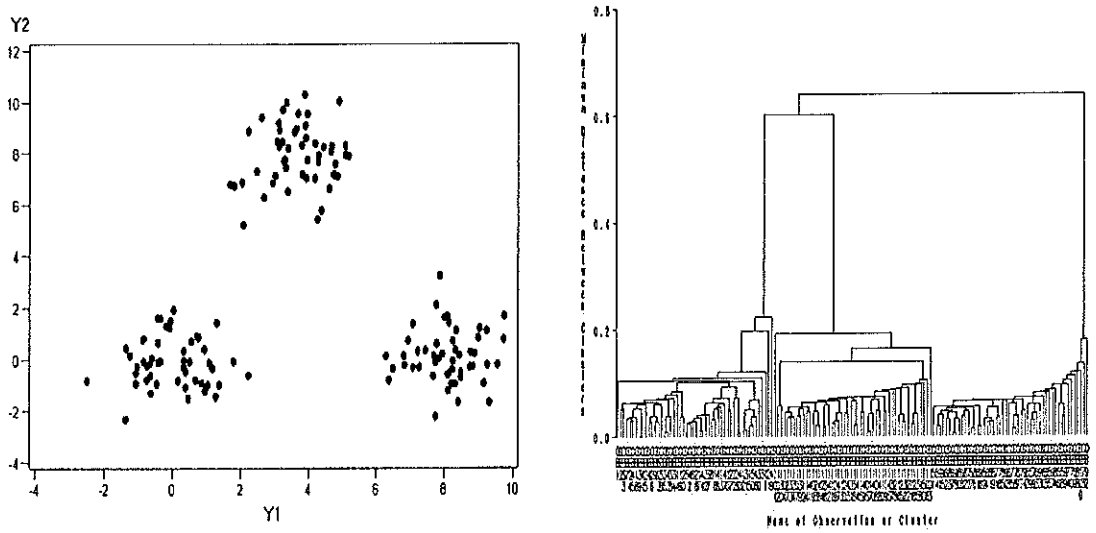


FIG. 1.2 – Trois classes intuitivement distinctes avec $n = 150$ et $p = 2$ (à gauche), output de la méthode C. A. H. du lien minimum (à droite)

Ces outputs sont appelés **dendrogrammes**. A tout palier est associée une quantité indiquée sur la gauche du dendrogramme. Il s'agit de la distance séparant les groupes de points réunis par ce même palier. L'algorithme ne forme enfin qu'une seule classe : le palier le plus haut E de la hiérarchie H. Lire un dendrogramme est chose aisée. En effet, n partitions y sont représentées, et toutes y sont donc qualitativement mesurées en termes d'isolation des classes composant les partitions en question.

Les algorithmes C. A. H. ne prennent pas en input d'entier k fixé, nombre de classes à distinguer. En ce qui concerne cet entier, c'est à l'utilisateur de déterminer sa valeur, soit en se basant sur l'allure du dendrogramme (à quel palier la coupure doit-elle être effectuée ?), soit au moyen des méthodes abordées dans le deuxième chapitre de ce mémoire.

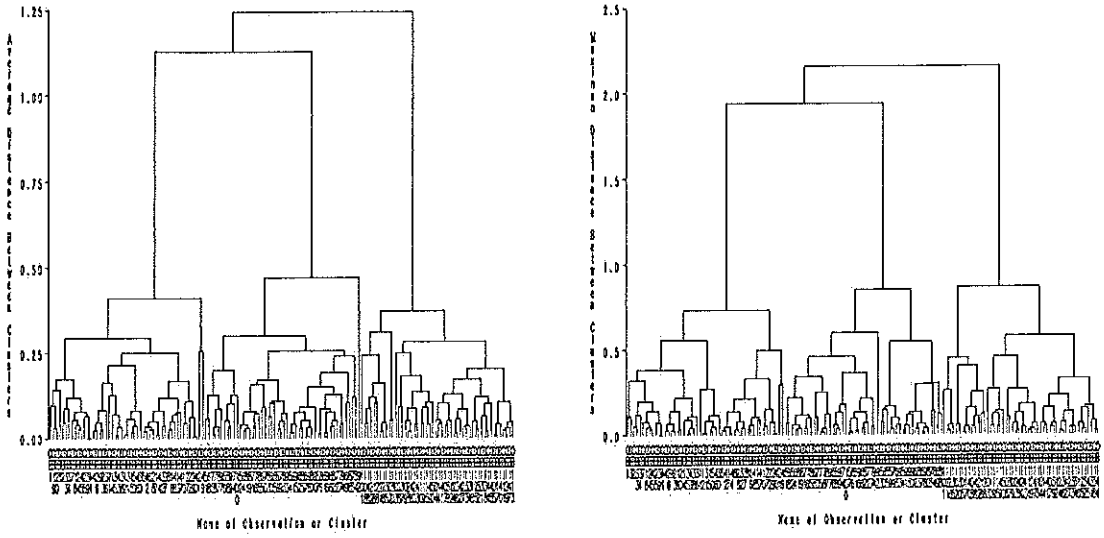


FIG. 1.3 – Outputs des méthodes C. A. H. du lien moyen (à gauche), du lien maximum (à droite)

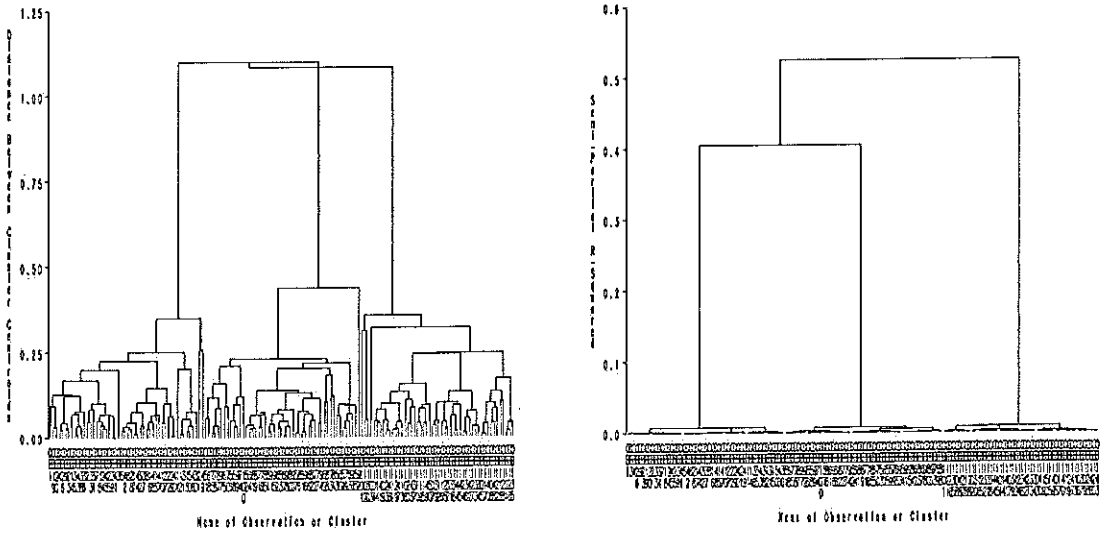


FIG. 1.4 – Outputs des méthodes C. A. H. des centroïdes (à gauche), de Ward (à droite)

Remarque

En observant d'assez près l'output de gauche de la figure 1.4, il est clairement visible que la dernière agrégation, celle des deux paliers de la hiérarchie aboutissant à E, s'est effectuée à un niveau d'agrégation moindre que l'agrégation précédente. Ce phénomène d'**inversion** est typique de la méthode C. A. H. des centroïdes.

Ces méthodes ont naturellement leurs avantages mais aussi leurs inconvénients, selon l'indice d'agrégation utilisé et le jeu de données analysé (effet de chaînage, biais pour classes hypersphériques, ...). Nous ne les évoquerons dans ce mémoire que lorsque nous y serons contraints, comme prévu précédemment dans ce document.

1.2 Méthode des centres mobiles

Voyons maintenant un second et dernier algorithme de classification : l'algorithme des centres mobiles [12], dont les étapes sont données ci-après.

Algorithme des centres mobiles (partitionnement de E en k classes)

1. Initialisation
   - Tirer k objets (des **centres**) au hasard parmi les n objets de E, construire une partition initiale $P^{(0)} = \{C_1^0, \ldots, C_k^0\}$ en attribuant aux n objets l'indice du centre le plus proche, au sens de la distance euclidienne d.
   - Calculer : $n_l := |C_l^0|$ et $g^{(0,l)}$, centroïde de la classe initiale C_l^0, $\forall\, l \in \{1, \ldots, k\}$.

2. Affectation
   $test := 0$.
   Pour $i = 1$ jusque $i = n$ faire :
   - noter s la classe de x_i,
   - déterminer l tel que
   $$l \;=\; \text{argmin}_j \; d^2(x_i, g^{(j)}), \qquad \text{où} \;\; j \in \{1, \ldots, k\},$$
   - si $l \neq s$ alors
   $$test := 1,$$
   $$C_l := C_l \cup \{x_i\},\; C_s := C_s \setminus \{x_i\},$$
   $$n_l := n_l + 1,\; n_s := n_s - 1.$$

3. Représentation
   Pour $l = 1$ jusque $l = k$ faire :
   calculer le centroïde de la nouvelle classe $g^{(l)}$.

4. Si $test \neq 0$ alors retourner à l'étape 2 d'affectation.

Remarques

1. L'algorithme des centres mobiles requiert l'entrée par l'utilisateur d'un nombre de classes fixé k, comme pour tout **algorithme de voisinage**. Cette information n'était pas nécessaire à l'exécution d'un algorithme C. A. H..

2. Il existe une variante de l'algorithme des centres mobiles : l'**algorithme des k-means** [12]. Il consiste à passer immédiatement à l'étape de représentation lors de chaque nouvelle indexation d'un objet à l'étape d'affectation. L'output dépend donc de l'ordre dans lequel sont analysés les objets, durant la boucle d'affectation. Une solution localement optimale de l'algorithme des centres mobiles est une solution localement optimale de l'algorithme des k-means et vice-versa.

3. La **méthode des nuées dynamiques** est une généralisation de la méthode des centres mobiles. En effet, dans la méthode des centres mobiles, le centroïde joue le rôle de noyau : le centroïde d'un groupe de points est le prototype choisi pour représenter au mieux ce groupe de points. L'algorithme des nuées dynamiques peut prendre en compte de multiples prototypes : le centroïde, un point de la classe, q points de la classe, une droite, etc... Cette généralisation implique évidemment des modifications dans les étapes d'affectation (quelle distance utiliser entre une classe et son prototype ?) et de représentation. Aucune de ces extensions ne sera employée ici.

Nous disposons dès à présent d'outils aidant à partitionner un jeu de données en k classes. Certes, aucune de ces méthodes n'est parfaite mais, une fois réunies, elles forment un panel efficace, capable de contrer la grande majorité des difficultés propres à certaines structures. Une question persiste : nous pouvons maintenant partitionner un ensemble d'objets E en k classes, mais restons dans le flou quant à la valeur réelle de l'entier k. Le chapitre suivant est particulièrement axé sur ce point.

Chapitre 2

Détermination théorique du nombre de classes

Ce second chapitre introductif a pour but de délimiter un cadre théorique concernant la détermination du nombre de classes que devrait contenir une partition *idéale* de l'ensemble d'objets E. Il s'agira aussi de rappeler brièvement certaines méthodes connues à ce jour.

2.1 Indices de validation internes, externes, relatifs

La **validation**, en classification, a pour but d'évaluer les outputs d'un algorithme de classification sur un jeu de données, de manière qualitative et objective.

Un **indice de validation** mesure l'adéquation d'une entité classificatoire, c'est-à-dire la manière dont cette entité apporte de l'information sur un jeu de données, ou tout simplement la capacité qu'a cette entité à refléter le caractère intrinsèque des données.

Les trois entités classificatoires à considérer ici sont les suivantes :

1. les hiérarchies, outputs d'algorithmes C. A. H. par exemple,
2. les partitions, outputs d'algorithmes de classification pour lesquels le paramètre k (le nombre de classes) a été spécifié ou décidé,
3. les classes, éléments de partitions.

Ces indices sont basés sur des **critères**, eux-mêmes divisés en trois types [17].

Les **critères externes** comparent un partitionnement, ou une partie de partitionnement, à des informations externes dont l'algorithme de classification n'a pas tenu compte durant son exécution. Nous pouvons, par exemple, considérer une partition réalisée par une équipe d'experts (caractérisée par une ou plusieurs colonne(s) supplémentaire(s) de labels dans le jeu de données).

Les **critères internes** comparent un partitionnement, ou une partie de partitionnement, avec les données originales. Comme critères internes, nous pouvons citer le critère de l'inertie intra-classe, ou encore le critère des hypervolumes, tous deux définis ultérieurement dans ce chapitre.

Quant aux **critères relatifs**, ce sont des critères mixtes par rapport aux précédents. Par exemple, ce sont eux qui comparent des outputs issus d'algorithmes de classification différents.

De nombreux indices sont déjà bien présents dans la littérature traitant de la classification. Nous ne nous intéresserons qu'à une petite partie d'entre eux : nous citerons certaines méthodes de **détermination du nombre de classes** ayant déjà prouvé leur efficacité.

2.2 Modèles nuls

Les modèles nuls sont des ensembles d'objets (générés ou non) **sans structure en classes**, c'est-à-dire composés d'une et une seule classe intuitive. Voyons deux modèles nuls, également très présents dans la littérature [11].

2.2.1 Le modèle de Poisson

Ce modèle a pour hypothèse que les objets peuvent être représentés par des points uniformément distribués dans un sous-ensemble A de $\mathbb{R}^p$ (l'**hypothèse d'uniformité**).

Les résultats issus de l'utilisation de critères basés sur ce modèle peuvent dépendre fortement du sous-ensemble A, et des justifications pour tel ou tel choix de A sont parfois très difficiles à fournir.

Une manière de régler le problème est de choisir A comme étant une dilatation de l'enveloppe convexe de tous les objets de l'ensemble d'objets original E.

Cette idée est issue du résultat suivant [22] :

si des points sont générés aléatoirement dans un sous-ensemble convexe (du plan), un estimateur non biaisé de la borne de ce sous-ensemble est une dilatation de l'enveloppe convexe de cet ensemble de points à partir de son centroïde.

Une approximation du coefficient de dilatation est la suivante [21] :

$$c = \sqrt{\frac{n}{n - V_n}},$$

où V_n est le nombre de points sur l'enveloppe convexe.

De tels modèles à **données influencées** ont été élaborés pour permettre la mise au point de tests qui peuvent examiner les écarts du jeu de données original par rapport à l'absence de structure en classes, comme nous le verrons ultérieurement.

Définition 2.2.1. *Soit $D \subset \mathbb{R}^p$. N est un processus ponctuel sur D si et seulement si N est une distribution aléatoire de points dans D telle que seuls un nombre fini d'entre eux sont répartis dans tout ensemble borné $A \subset D$, i. e.*

$$A \subset D \text{ borné} \implies N(A) < \infty.$$

Soit maintenant $m(\cdot)$ la **mesure de Lebesgue multidimensionnelle**. C'est la *longueur* en dimension 1, l'*aire* en dimension 2, le *volume* en dimension 3, l'*hypervolume* en dimension p.

Définition 2.2.2. *N est un processus de Poisson homogène (P. P. H.) d'intensité $q > 0$ sur $D \subset \mathbb{R}^p$ $(0 < m(D) < +\infty)$ s'il s'agit d'un processus de comptage sur D vérifiant les assertions suivantes :*

1. *les variables aléatoires $N(A_i)$ qui comptent les nombres de points dans des sous-ensembles disjoints $A_i \subset D$ sont indépendantes,*

2. *$\forall\, A \subset D$, $\forall\, k \geq 0$,*

$$P(N(A) = k) = e^{-q.m(A)} \frac{(q.m(A))^k}{k!},$$

la variable aléatoire $N(A)$ suit une loi de Poisson de paramètre $q.m(A)$.

L'appellation du modèle de cette sous-section vient de la proposition suivante :

Proposition 2.2.1. *Si N est un P. P. H. sur D et si $N(A) = n$ est fini $(A \subset D)$, alors les n points sont distribués* **uniformément** *dans A.*

Ce modèle est le fondement de la **méthode polythétique divisive des hypervolumes**, algorithme de classification non abordé dans ce mémoire.

2.2.2 Le modèle normal

Ce modèle a pour hypothèse que la distribution jointe des variables caractérisant les objets est normale. Autrement dit, les p variables descriptives forment un vecteur gaussien, de moyenne $\mu \in \mathbb{R}^p$ et de dispersion Σ.

Les points du modèle sont donc supposés plus proches les uns des autres lorsqu'ils sont près du *centre* de la classe et plus éloignés lorsqu'ils sont près de sa *borne*.

Il s'avérera nécessaire de reparler de ces modèles nuls dans le chapitre suivant, en particulier du modèle de Poisson sur lequel est bâtie la méthode de la Gap Statistic.

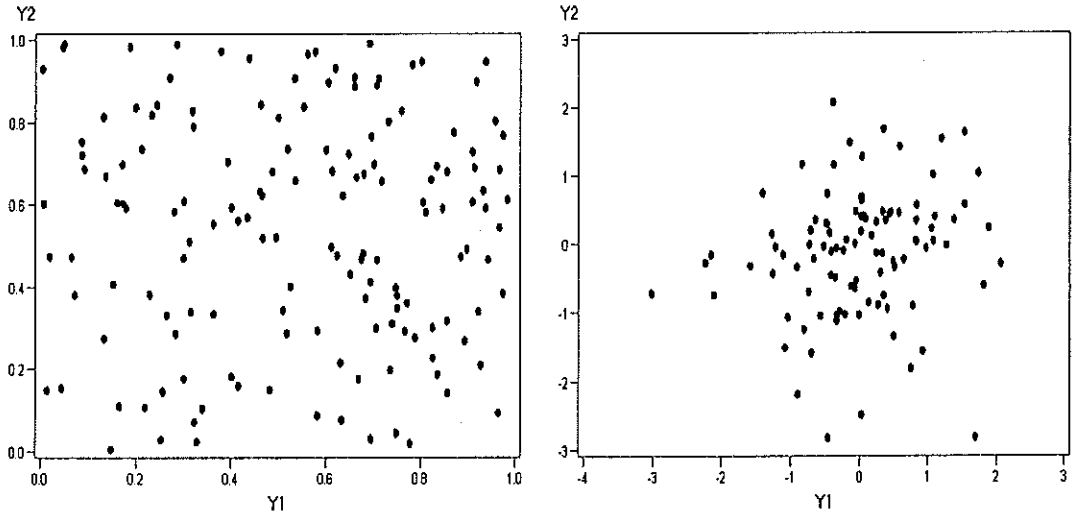


FIG. 2.1 – Simulations d'un modèle de Poisson (à gauche), d'un modèle normal (à droite)

2.3 Méthodes globales et locales

Dans la majorité des études de jeux de données en classification automatique, l'**information externe** n'est pas disponible. Il est donc devenu monnaie courante de chercher des réponses aux deux questions susceptibles d'émerger après partitionnement de l'ensemble E :

- Une partition en k classes, output d'un algorithme de partitionnement, comprend-elle des classes **cohésives** et **isolées** ?

- Pour quelle(s) valeur(s) de k est-il raisonnable de considérer le partitionnement comme **significatif** et **utile** ?

Ces deux questions s'adressent respectivement aux tests de validation internes et relatifs liés au partitionnement. La seconde influencera la première. En effet, réaliser un partitionnement de E en k classes (où k est une valeur non intuitive) semble peu cohérent. Des procédures qui cherchent à déterminer l'unique valeur de k, du moins la plus appropriée, sont généralement basées sur des **règles d'arrêt**.

Notons avant tout qu'une théorie de détermination du nombre de classes ne dépend pas de la méthode de partitionnement sous-jacente. Ainsi, quelle que soit la partition en output, les étapes du test choisi appliqueront les mêmes opérations sur les composantes de cette partition.

Ces règles d'arrêt peuvent être **locales** ou **globales**.

- Les **règles d'arrêt globales** utilisent toute l'information contenue dans une partition en k classes, pour chaque valeur de k. La valeur de k pour laquelle un critère spécifique est satisfait est alors identifiée. La plupart des critères sont basés sur la maximisation ou la minimisation de ces mesures et comparent les comportements internes (la cohésion) et externes (l'isolation) des classes. Il se peut que ces mesures

ne soient pas définies en $k = 1$. C'est un inconvénient important puisqu'il ne nous permet pas de dire si l'ensemble n'est formé que d'une et une seule classe, de vérifier l'absence de structure en classes.

- Les **règles d'arrêt locales** sont basées sur des tests examinant si une classe doit être divisée en deux ou non. Contrairement aux règles globales donc, elles doivent être appliquées à des outputs de méthodes de partitionnement hiérarchiques (descendantes ou ascendantes). Elles n'utilisent donc également qu'une partie des données à la fois (un palier de dendrogramme, par exemple). Il est parfois nécessaire de spécifier un seuil de confiance, propre au test en question, dont la valeur peut influencer grandement les propriétés de la règle d'arrêt, ainsi que sa **décision**.

2.4 Des méthodes de détermination du nombre de classes

Voyons maintenant une série de méthodes de détermination du nombre de classes intuitives, série très courte en comparaison de toutes les méthodes développées dans la littérature. Ces méthodes ne seront pas toutes utilisées dans ce document.

Rappelons avant tout le problème de détermination du nombre de classes.
Soit l'ensemble $E = \{x_1, \ldots, x_n\}$. Il est attribué à cet ensemble p descripteurs $Y_1, \ldots, Y_p$.
Le but est de déterminer l'entier positif non nul k pour lequel une partition $P^\star = \{C_1^\star, \ldots, C_k^\star\}$ de E semble idéale.

Définition 2.4.1. *Nous noterons par g le centroïde de l'ensemble E tout entier. Soit une classe C_l, élément d'une partition P de E (en k classes). Nous noterons par $g^{(l)}$ le centroïde de la classe C_l.*

1. *L'inertie intra-classe d'une partition entière P de l'ensemble E (en k classes) est notée $W(P,k)$:*
$$W(P,k) = \sum_{l=1}^{k} I(C_l) = \sum_{l=1}^{k} \sum_{x_i \in C_l} d^2(x_i, g^{(l)}).$$
2. *L'inertie inter-classe d'une partition P de l'ensemble E (en k classes) est notée $B(P,k)$:*
$$B(P,k) = \sum_{l=1}^{k} n_l \, d^2(g^{(l)}, g),$$
où n_l est le cardinal de C_l : $n_l = |C_l|$.
3. *L'inertie totale d'une partition P de E (en k classes) est notée $T(P,k)$:*
$$T(P,k) = I(E) = \sum_{i=1}^{n} d^2(x_i, g).$$

Nous noterons par $R^2(P,k)$ le rapport
$$\frac{B(P,k)}{T(P,k)} \in [0,1].$$

Si ce coefficient R^2 est proche de 1 (ou si $1 - R^2$ est proche de 0), la partition est dite **justifiée au sens de l'inertie.**

Proposition 2.4.1. *[12] Pour toute partition P de l'ensemble E en k classes,*

$$T(P,k) = W(P,k) + B(P,k).$$

Par conséquent,

minimiser l'inertie intra-classe dans un souci de compacité et de cohésion des classes est équivalent **à maximiser l'inertie inter-classe** dans un souci d'isolation des classes de la partition.

2.4.1 La méthode du coude

Cette méthode géométrique consiste à tracer la valeur d'un critère $\mathcal{W}(P,k)$, en fonction de l'entier strictement positif k. La présence d'un **coude** indique le nombre optimal de classes naturelles, au sens de la partition P. Ce critère peut prendre de multiples formes :

- $\mathcal{W}(P,k) = 1 - R^2(P,k)$,
- le critère de l'inertie intra-classe, équivalent au premier, à une constante près : $\mathcal{W}(P,k) = W(P,k)$,
- le critère des hypervolumes [13] : $\mathcal{W}(P,k) = \sum_{l=1}^{k} m(H(C_l)))$,

  où $m(\cdot)$ est la mesure de Lebesgue multidimensionnelle, évoquée précédemment.

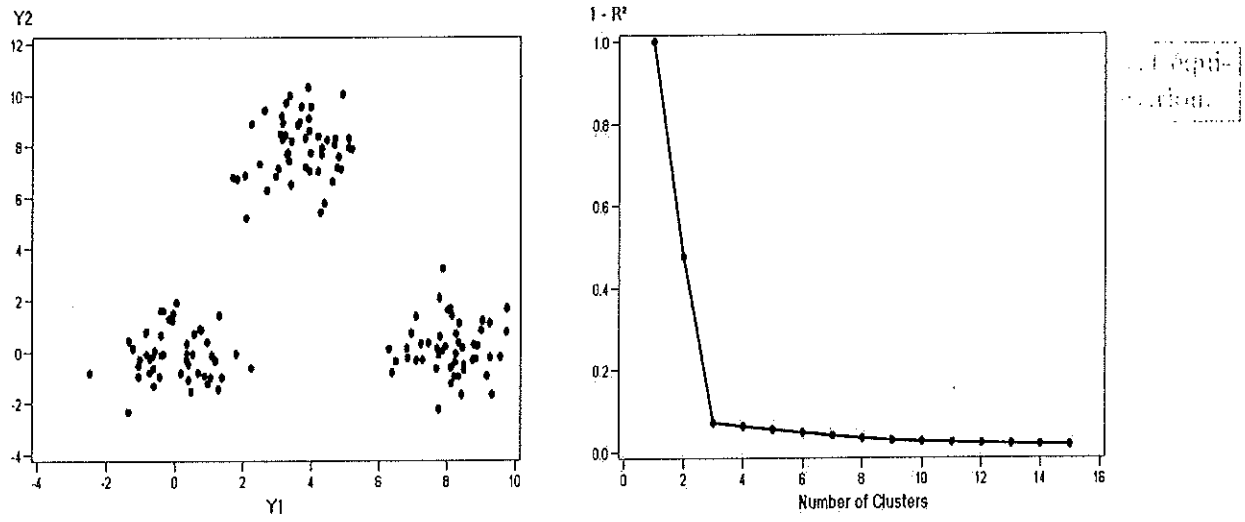


FIG. 2.2 – Illustration de la méthode du coude au jeu de données de gauche (3 classes distinctes) sur un output de la méthode C. A. H. de Ward

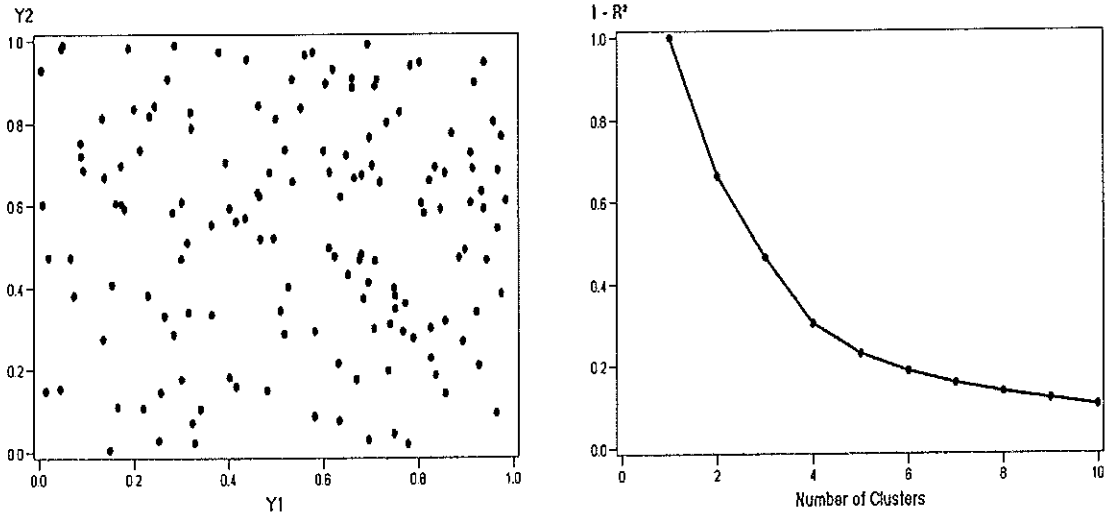


FIG. 2.3 – Illustration de la méthode du coude au jeu de données de gauche (pas de structure en classes) sur un output de la méthode C. A. H. de Ward

Remarque

Le coude est bien marqué en $k = 3$ dans le premier exemple (figure 2.2). Par contre, dans le second exemple, généré selon un modèle uniforme (figure 2.3), aucune valeur de k n'est décisive quant à la formation d'une cassure en coude de la courbe.

Notons que **les trois critères définis précédemment sont décroissants**, en fonction de k.

Malgré sa simplicité, cette méthode reste basée sur l'intuition, puisque géométrique. Elle est par conséquent peu rigoureuse.

2.4.2 Les six meilleures méthodes de Milligan & Cooper

Les méthodes reprises dans cette sous-section ne se basent plus que sur l'unique information que fournit la matrice carrée d'ordre n symétrique reprenant les n^2 distances (euclidiennes) entre chaque paire d'objets de E. Elles sont très souvent reprises dans la littérature ??, pour avoir prouvé leur efficacité, particulièrement sur les jeux de données aux classes hypersphériques séparées.

La méthode de Calinski et Harabasz [7]

Il s'agit d'une méthode globale.

Définissons l'indice

$$CH(P,k) := \frac{B(P,k)/(k-1)}{W(P,k)/(n-k)},$$

où $B(P,k)$ et $W(P,k)$ sont les inerties inter-classe et intra-classe de la partition P en k classes sortie par l'algorithme de classification utilisé.

La valeur de $k^\star$, le *bon* nombre de classes naturelles, sera la valeur en laquelle l'indice sera maximum relatif ou absolu ou aura un écart important par rapport à la valeur précédente.

Remarquons que l'indice n'est pas défini pour $k = 1$.

La méthode de Duda et Hart [9]

Il s'agit d'une méthode locale, à appliquer à chaque palier d'un output d'un algorithme de classification hiérarchique, à partir du plus haut de ses paliers : celui qui réunit les deux dernières classes qui formeront l'ensemble entier E.

Soit p, le palier de la hiérarchie auquel la classe C_p est formée par la réunion des deux sous-classes $C_{p,1}$ et $C_{p,2}$:

$$C_p = C_{p,1} \cup C_{p,2}.$$

Les deux hypothèses du test sont

$$\begin{cases} H_0 : \text{les points de } C_p \text{ sont issus d'une population normale de moyenne } \mu \\ \qquad\quad \text{et de matrice de dispersion } \sigma^2 I, \\ H_1 : \text{les points de } C_p \text{ sont issus de deux populations normales.} \end{cases}$$

Définissons la statistique suivante :

$$DH := \frac{\left[-\frac{I(C_{p,1})+I(C_{p,2})}{I(C_p)} + 1 - \frac{2}{\pi p}\right]}{\sqrt{\frac{2\left(1-\frac{8}{\pi^2 p}\right)}{np}}}.$$

H_0 sera rejetée, au niveau α, si $DH \geq z_{1-\alpha}$, avec $z_{1-\alpha} = 3.20$ (ou encore $z_{1-\alpha} = 4.00$, en fonction de la littérature suivie).

Si p_0 est le premier palier pour lequel l'hypothèse H_0 est rejetée, et si p_0 est le l^{eme} palier analysé, alors le nombre de classes sera $k^\star = l + 1$.

Remarquons que, comme précédemment, cette méthode ne permet pas de vérifier l'absence de structure en classes, ou plus simplement l'existence d'une classe unique.

La méthode de l'indice C [16]

Il s'agit d'une méthode globale.

Posons

$$\Lambda(k) := \sum_{l=1}^{k} \sum_{x_i, x_j \in C_l \ i<j} d(x_i, x_j).$$

Normalisons

$$C(k) := \frac{\Lambda(k) - \min \Lambda(k)}{\max \Lambda(k) - \min \Lambda(k)}.$$

Posons $r(k) := \sum_{l=1}^{k} C^2_{|C_l|}$. Il s'agit du nombre de dissimilarités entre paires d'individus issues de mêmes classes, considérées deux à deux.

$\min \Lambda(k)$ est la somme des r plus petites dissimilarités entre les paires d'individus de tout l'ensemble E,

$\max \Lambda(k)$ est la somme des r plus grandes dissimilarités entre les paires d'individus de tout l'ensemble E.

Par conséquent, l'indice $C(k)$ est compris dans l'intervalle $[0, 1]$.

Le nombre de classes naturelles est indiqué par la valeur minimale de l'indice. Idéalement, il vaut 0.

Cette méthode n'est pas capable de décerner l'absence de structure en classes.

Méthode de l'indice γ [3]

Il s'agit d'une méthode globale.

Posons

$$T_{ij} = T(x_i, x_j) := \begin{cases} 0 & \text{si les objets } x_i \text{ et } x_j \text{ appartiennent à la même classe,} \\ 1 & \text{sinon.} \end{cases}$$

Si $T_{ij} = 0$, posons

$$n_{ij} = n(x_i, x_j) := card \quad \{(x_r, x_s) : T_{rs} = 1, \quad d(x_r, x_s) < d(x_i, x_j)\}.$$

n_{ij} est donc le nombre de couples d'objets qui n'appartiennent pas à la même classe et qui sont plus proches, au sens de d, que ne le sont x_i et x_j.

Posons

$$\alpha := \frac{\sum_{i<j} n_{ij}}{\max \sum_{i<j} n_{ij}}.$$

Le maximum au dénominateur est pris sur toutes les partitions possibles en k classes en gardant le même nombre d'objets par classe.

Les deux sommes sont réalisées sur les paires d'objets $\{x_i, x_j\}$ qui appartiennent à la même classe.

α est ainsi une proportion de paires d'objets *mal classés*.

Posons maintenant

$$\gamma := 1 - 2\alpha \quad \in [-1, 1].$$

- si aucun objet n'est *mal classé*, alors $\alpha = 0$ et donc $\gamma = 1$,

- par contre, si beaucoup d'objets semblent *mal classés*, alors la valeur de γ aura tendance à décroitre vers -1 ($\alpha \simeq 1$).

Si c'est sur la partition en $k^\star$ classes que l'indice γ est maximal, alors $k^\star$ sera le nombre de classes recherché.

Comme les méthodes précédentes, cette méthode n'est pas capable de décerner l'absence de structure en classes.

La méthode de Beale [4]

Cette méthode est, comme la méthode de Duda & Hart, basée sur un test portant sur la fusion de deux classes C_1 et C_2, à chaque palier d'une hiérarchie. Le test de Beale est donc une méthode locale. Les hypothèses à considérer sont les suivantes :

$$\begin{cases} H_0 & : \text{ la fusion est justifiée,} \\ H_1 & : \text{ la fusion n'est pas justifiée, les deux classes doivent rester distinctes.} \end{cases}$$

Posons

$$W := \frac{\frac{I(C_1 \cup C_2) - I(C_1) - I(C_2)}{I(C_1) + I(C_2)}}{\left(\left(\frac{n-1}{n-2}\right) 2^{\frac{2}{p}} - 1\right)}.$$

Sous H_0, $W \approx F(p, (n-2)p)$.

La règle de décision est donc la suivante :

H_0 est rejetée, au niveau α, si $W \geq k_\alpha$, où k_α est tel que

$$P_{H_0}(W \geq k_\alpha) = \alpha.$$

Si k_0 est la première valeur conduisant au rejet de la fusion des deux groupes, le test de Beale indique que le bon nombre de classes est $k_0 + 1$.

La méthode du Critère de Classification Cubique [23]

Il s'agit d'une méthode globale, faisant intervenir la génération d'un **modèle uniforme**. Elle est le dernier procédé repris par le top 6 de Milligan et Cooper.

Posons

$$CCC := \log\left[\frac{\frac{1-E(R^2)}{1-R^2}}{\frac{\sqrt{np/2}}{(0.001+E(R^2))^{1.2}}}\right],$$

où

- n est le nombre d'objets de E,
- p est l'estimation optimale de la **dimensionnalité**, déterminée par une analyse en composantes principales,

- $E(R^2)$ est calculé en supposant que les données sont générées selon une **distribution uniforme multidimensionnelle.**

La valeur de l'indice correspondant au pic maximal indique le nombre de classes k^* à retenir.

Un crédit respectable sera attribué à la partition si $CCC > 2$ (ou 3, selon la littérature adoptée). Remarquons que la méthode est construite de sorte qu'elle puisse vérifier l'absence de structure en classes. C'est d'ailleurs l'unique méthode du top 6 de Milligan et Cooper dotée de cette caractéristique.

2.4.3 Une méthode basée sur le critère des hypervolumes

Méthode du test des hypervolumes [13]

La méthode décrite ici présente l'avantage de reposer sur un modèle statistique construit.

Soit $X_1, \ldots, X_n$ la réalisation d'un processus de Poisson homogène dans t domaines **convexes disjoints** d'un espace euclidien à p dimensions. Le but est ici de tester si une subdivision en k classes est significativement meilleure qu'une subdivision en $k-1$ classes :

$$\begin{cases} H_0 & : \quad t = k, \\ H_1 & : \quad t = k-1. \end{cases}$$

Posons :

- $C = \{C_1, C_2, \ldots, C_k\}$ la partition optimale de la réalisation en k classes,
- $B = \{B_1, B_2, \ldots, B_{k-1}\}$ la partition optimale de la réalisation en $k-1$ classes.

Associons à ce modèle une fonction de vraisemblance :

$$L(D; x_1, \ldots, x_n) = \frac{1}{(m(D))^n} \mathbb{1}_D(H(x_1, \ldots, x_n)),$$

où $m(\cdot)$ est donc la mesure de Lebesgue multidimensionnelle.

L'artifice utilisé ici est le test du quotient de vraisemblance généralisé :

$$\begin{aligned} \lambda(x_1, \ldots, x_n) &:= \frac{\sup_B L(B; x_1, \ldots, x_n)}{\sup_C L(C; x_1, \ldots, x_n)} \\ &= \frac{\frac{1}{\left(\sum_{l=1}^{k-1} m(H(B_l))\right)^n}}{\frac{1}{\left(\sum_{l=1}^{k} m(H(C_l))\right)^n}} \\ &= \left(\frac{W(P^*, k)}{W(P^*, k-1)}\right)^n \\ &=: S^n, \end{aligned}$$

avec

$$S = \left(\frac{W(P^*, k)}{W(P^*, k-1)}\right)$$

et, cette fois, W est considéré comme critère des hypervolumes :

$$W(P,k) = \sum_{l=1}^{k} m(H(C_l)).$$

En réalité, la région critique est de la forme suivante :

$$RC = \{\lambda = S^n > k\}$$

$$= \{S > k' := k^{\frac{1}{n}}\}.$$

Malgré le fait que la distribution sous H_0 de la statistique S soit inconnue, cette même statistique présente la propriété logique suivante :

$$S \in [0,1].$$

Empiriquement, H_0 sera rejetée si S prend de grandes valeurs, c'est à dire si $S \simeq 1$.

Si k_0 est la première valeur pour laquelle H_0 est rejetée, alors le jeu de données sera supposé divisé en $k_0 - 1$ classes.

2.4.4 Comparaison succincte des méthodes

Prenons maintenant en compte le tableau suivant [12] :

	C & H	D & H	C	Γ	Beale	CCC	Hypervolumes
Classes hyperellipsoïdales	X	X	X	X	X	X	
Données sans structure en classes	X	*qu.*					
Classes de tailles inégales éloignées					X	X	
Classes de tailles inégales proches		X			X	X	
Classes non séparables par un hyperplan		X	X	X	X	X	
Classes non convexes	X	X	X	X	X	X	X

Cette grille est le **tableau de comparaison des méthodes de détermination du nombre de classes**, reprenant toutes les méthodes exposées précédemment ainsi que les différents types de jeux de données rencontrés dans la littérature.
Les "X" de la grille sont les problèmes potentiels, tandis que *qu.* signifie que le résultat dépend du seuil de signification associé au test. Ce test est ici celui de Duda & Hart. Ses résultats sur des données sans structure en classes varient donc en fonction du quantile $z_{1-\alpha}$ choisi (3.20 ou 4).

Remarque

Nous avons évoqué les tests de Beale et des hypervolumes dans ce chapitre essentiellement dans un but de comparaison théorique avec la méthode traitée dans ce mémoire, décrite à partir du chapitre suivant. Les statistiques de ces deux tests ne seront pas calculées dans les chapitre concernant les applications classiques et symboliques.

Chapitre 3

La méthode de la Gap Statistic

En 2000, Tibshirani, Walther et Hastie (Université de Stanford, Californie) ont mis au point une méthode de détermination du nombre de classes naturelles au sein d'une base de données quantitatives classiques : la **méthode de la Gap Statistic** [25]. Ce chapitre a pour but l'étude de cette méthode ainsi que sa justification théorique.

3.1 Introduction

Le problème de classification est invariable : nous disposons de E, ensemble de n objets x_i caractérisés par p descripteurs Y_j à valeurs dans $\mathcal{Y}_j \subset \mathbb{R}$. Comment réaliser un partitionnement cohérent de cet ensemble E en k classes ?

Quant au problème de détermination du nombre de classes étudié au chapitre précédent, il s'agit de trouver une valeur *significative* $k^\star$, cardinal naturel de la meilleure partition.

Notons W_k l'inertie intra-classe d'une partition en k classes de E, $P = \{C_1, \ldots, C_k\}$, output d'un algorithme $\mathcal{A}$ de partitionnement fixé. Remarquons qu'il s'agit d'une notation allégée par rapport au chapitre précédent.

W_k décroît de manière monotone en fonction de k ; mais à partir d'un certain $k^\star$, un effet d'**écrasement de la courbe** est apparent (voir figure 2.2, à la sous-section du chapitre précédent consacrée à la méthode du coude). C'est cet entier positif $k^\star$ que la méthode suivante nous aidera à déterminer, cette fois de manière non seulement intuitive mais mathématique.

Soit

$$I(C_l) = \sum_{x_i \in C_l} d^2(x_i, g^{(l)}),$$

l'inertie de la classe C_l, élément d'une partition P.

Voyons comment reformuler de manière pratique cette définition.

Proposition 3.1.1.

$$I(C_l) = \frac{1}{2\,|C_l|} \sum_{x_i \in C_l} \sum_{x_j \in C_l} d^2(x_i, x_j).$$

Preuve. [12]

$$\begin{aligned}
\textstyle\sum_{x_i \in C_l} \sum_{x_j \in C_l} d^2(x_i, x_j) &= \textstyle\sum_i \sum_j (x_i - x_j)'(x_i - x_j) \\
&= \textstyle\sum_i \sum_j (x_i - g^{(l)} + g^{(l)} - x_j)'(x_i - g^{(l)} + g^{(l)} - x_j) \\
&= \textstyle\sum_{i,j} (x_i - g^{(l)})'(x_i - g^{(l)}) + \sum_{i,j} (x_i - g^{(l)})'(g^{(l)} - x_j) \\
&\quad + \textstyle\sum_{i,j} (g^{(l)} - x_j)'(x_i - g^{(l)}) + \sum_{i,j} (g^{(l)} - x_j)'(g^{(l)} - x_j) \\
&= n_l \textstyle\sum_i d^2(x_i, g^{(l)}) + 0 + 0 + n_l \sum_j d^2(x_j, g^{(l)}) \\
&= 2\,|C_l|\; I(C_l).
\end{aligned}$$

Corollaire 3.1.1.

$$T := I(E) = \frac{1}{2\,n} \sum_{i=1}^{n} \sum_{j=1}^{n} d^2(x_i, x_j).$$

Cette reformulation a pour avantage de ne pas faire intervenir le calcul des centroïdes $g^{(l)}$ et g. Elle sera utilisée dans les trois codes de programmation attachés à ce mémoire, commentés en annexe.

L'idée de la méthode est de normaliser la quantité $\log(W_k)$ en la comparant à son espérance sous l'hypothèse d'une **distribution de référence issue d'un modèle nul (D. R. M. N.)**.

L'estimation du nombre de classes naturelles serait ainsi la première valeur de k pour laquelle la courbe $\log(W_k)$ descend au plus bas, au-dessous de la courbe associée à la valeur $\log(W_k^\star)$, définie ci-après.

Définissons donc

$$Gap(k) := \log(W_k^\star) - \log(W_k),$$

où $\log(W_k^\star)$ est le logarithme de l'inertie intra-classe de la partition rendue par $\mathcal{A}$ du jeu de données généré **sous une D. R. M. N.**.

Notre estimation $k^\star$ sera la "première valeur k maximisant" la quantité $Gap(k)$, l'écart, en termes d'inertie, entre les deux jeux de données. Le concept de maximisation sera décrit ultérieurement dans ce chapitre, dans la section consacrée à l'algorithme proprement dit.

Cette méthode est très générale car applicable aux partitions fournies par tout algorithme de classification, au même titre que les méthodes décrites au chapitre précédent. Justifions maintenant l'utilisation des concepts précédemment énoncés.

3.2 La distribution de référence

La méthode va en premier lieu considérer l'hypothèse que les objets de E sont distribués selon un modèle nul ($k^* = 1$). Elle décidera ensuite si cette hypothèse doit être rejetée en faveur d'un modèle en $k^* > 1$ classes ou non :

$$\begin{cases} H_0 & : \quad k^* = 1, \\ H_1 & : \quad k^* > 1. \end{cases}$$

Cette vérification doit être réalisée pour tout $k > 1$, jusqu'à ce qu'un rejet du modèle nul soit effectué. En pratique, si le rejet n'a pas été effectué après un grand nombre J de tests par la règle d'arrêt, l'algorithme stoppe et l'hypothèse initiale reste à considérer.

Un modèle nul est caractérisé par le fait que sa distribution peut être approximée par une fonction dite *log-concave*, c'est-à-dire par une densité de la forme $\exp\{\psi(x)\}$, où ψ est fonction concave. Un exemple classique est naturellement la distribution normale centrée réduite, où

$$\psi(x) = -\frac{1}{2}\|x\|^2.$$

Notons par $\mathcal{S}^p$ l'ensemble des vecteurs aléatoires sur $\mathbb{R}^p$ dont les distributions génèrent des modèles nuls (dans $\mathbb{R}^p$ donc).

Pour voir quel modèle nul X^* choisir, quel vecteur aléatoire de $\mathcal{S}^p$ dont la distribution de référence nous semble la plus intéressante, considérons d'abord l'expression suivante, dans un cadre de partitionnement en k classes réalisé sur deux modèles nuls de $\mathcal{S}_p$.

Avec X^* et $X \in \mathcal{S}^p$ donc,

$$g(k) := \log\left(\frac{MSE_{X^*}(k)}{MSE_{X^*}(1)}\right) - \log\left(\frac{MSE_X(k)}{MSE_X(1)}\right),$$

de telle sorte que $g(1) = 0$, où

$$MSE_X(k) := E\left[\min_{\mu \in A_k} \|X - \mu\|^2\right],$$

avec $A_k \subset \mathbb{R}^p$ et $|A_k| = k$.

$MSE_X(k)$ correspond à l'inertie intra-classe théorique normalisée du jeu de données généré selon la distribution de X, partitionné en k classes. Par conséquent, la quantité

$$\frac{MSE_X(k)}{MSE_X(1)}$$

correspond à la valeur théorique de la quantité précédemment définie :

$$1 - R^2.$$

Ainsi, rechercher la D. R. M. N. sur $X^\star$ telle que $g(k) \leq 0$ pour tout $X \in \mathcal{S}^p$ et pour tout $k \geq 1$ est équivalent à rechercher la distribution dont le modèle nul (partitionné) est le mieux expliqué, en termes de variance.

Le théorème suivant règle cette question dans le cas univarié. La distribution de référence recherchée est en réalité la distribution uniforme sur l'intervalle unitaire $[0, 1]$.

Théorème 3.2.1. *Pour tout $k \geq 1$, avec $p = 1$,*

$$\inf_{X \in \mathcal{S}^p} \frac{MSE_X(k)}{MSE_X(1)} = \frac{MSE_U(k)}{MSE_U(1)}. \tag{3.1}$$

La preuve de ce résultat, disponible dans l'article de référence [25] mais très technique, ne sera pas réalisée ici.

En d'autres mots, parmi toutes les distributions univariées de modèles nuls, la **distribution uniforme** est la plus à même à générer des jeux de données dont les quantités théoriques $1 - R^2(P, k)$ sont les plus **petites**, selon le nombre de classes k. Le modèle uniforme est donc, dans le cas univarié, le moins bon des modèles nuls, en ce sens qu'il est celui pour lequel l'assertion $k^\star = 1$ est la moins facilement vérifiable. Parmi tous les modèles nuls, il est donc celui dont la variance est la mieux expliquée.

La *Gap Statistic* a, rappelons-le, pour but de trouver le nombre de classes pour lequel la classification sur le jeu de données original s'écarte le plus de la classification sur le modèle nul uniforme, en termes d'inerties. Ainsi, la variance d'un jeu de données E sera comparée à la variance d'un modèle nul uniforme, parce que ce modèle uniforme est le plus à même, parmi tous les modèles nuls, à confondre les deux variances.

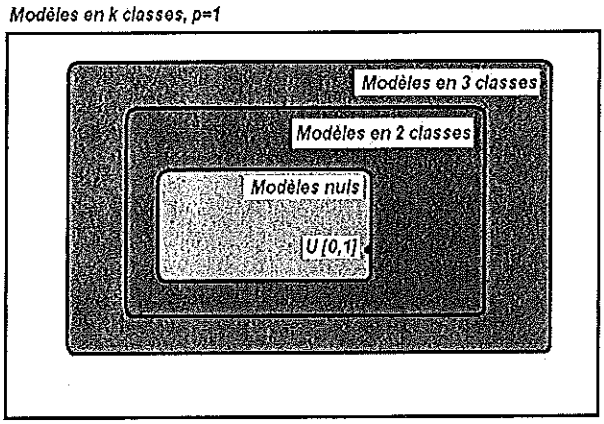


FIG. 3.1 – Le modèle nul uniforme est à la frontière des modèles non nuls, en termes d'explication de sa variance

Aussi, remarquons que

$$\frac{MSE_U(k)}{MSE_U(1)} = \frac{\frac{\left(\frac{1}{k}-0\right)^2}{12}}{\frac{(1-0)^2}{12}} = \frac{1}{k^2}.$$

Ce résultat nous renvoie à la méthode de détermination du nombre de classes de Krzanowski et Lai [19], nous suggérant de comparer successivement les valeurs des indices $W_k k^{\frac{2}{p}}$. Cette méthode est reprise ici.

Soit W_k l'inertie intra-classe de la partition en k classes, output d'un algorithme de classification donné.

Posons

$$DIFF(k) := (k-1)^{2/p} W_{k-1} - k^{2/p} W_k.$$

Choisissons $k^\star$ comme étant la variable maximisant

$$KL(k) := \left| \frac{DIFF(k)}{DIFF(k+1)} \right|.$$

Notons que cette méthode est globale, mais qu'elle ne permet pas de vérifier l'absence de structure en classes.

Le théorème suivant (dont la preuve n'est pas reprise dans ce mémoire mais réside dans l'article de référence [25]) nous montre qu'une approche, similaire à celle précédemment développée dans cette section, n'est malheureusement pas directement réalisable dans le cas multivarié ($p > 1$) :

Théorème 3.2.2. *Si $p > 1$, alors aucune distribution uniforme de S^p ne vérifie (3.1), à moins que son support de $\mathbb{R}^p$ ne soit réduit à un intervalle.*

Ce résultat nous montre que, dans le cas multivarié, nous ne serons pas en mesure de choisir une distribution de référence générale et réellement utile.

La prochaine section nous conseille quant à la manière d'utiliser ces deux théorèmes dans l'implémentation proprement dite de la méthode. Il s'agira, pour un ensemble E décrit par $p > 1$ descripteurs, de considérer séparément les p intervalles d'observation, et de générer uniformément sur chacun d'eux n points de $\mathbb{R}$, selon le théorème 3.2.1. Groupant ces p séries de points générés selon une dimension, nous obtiendrons un processus de Poisson homogène généré sur l'hypervolume d'observation du jeu de données original.

3.3 Implémentation de la méthode

Avant de nous lancer dans la génération d'un modèle nul adéquat, voyons ce qu'est la **bounding box** d'un jeu de données, concept également nommé **hypervolume d'observation**. La bounding box d'un jeu de données E, caractérisé par p descripteurs, est l'hyperrectangle dans $\mathbb{R}^p$ dont les arêtes sont les espaces d'observations $\mathcal{Y}_j$ des p descripteurs.

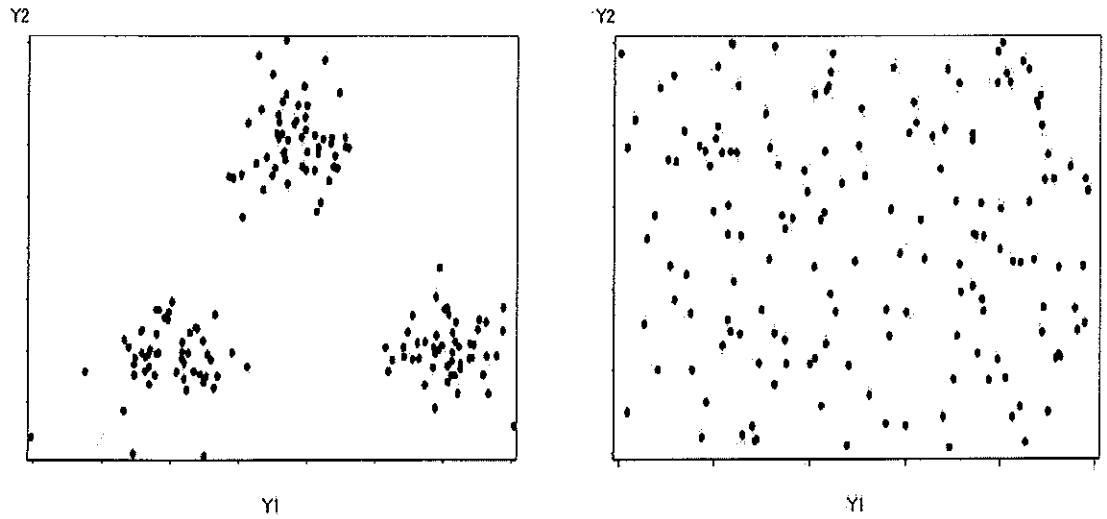


FIG. 3.2 – Calcul de la bounding box (en trait plein) d'un jeu de données décrit par deux variables (à gauche) et génération d'un processus de Poisson homogène dans cette même bounding box (à droite)

L'opération représentée par la figure 3.2 correspond à la deuxième étape de l'algorithme général suivant.

Algorithme déterminant la nombre de classes au moyen de la Gap Statistic

1. **Partitionner** le jeu de données original au moyen de l'algorithme $\mathcal{A}$ de classification choisi, en $k = 1, \ldots, K$ classes, et **stocker** en mémoire les inerties intra-classes W_k, $k = 1, \ldots, K$.
2. **Créer** B jeux de données de référence, par génération de B processus de Poisson (homogènes) selon les mêmes bounding box et taille que le jeu de données original, et **partitionner** chacun d'eux, **stocker** les inerties intra-classes associées $W^*_{k,b}$, $b = 1, \ldots, B$, $k = 1, \ldots, K$.
3. **Calculer** la Gap Statistic (estimée) :

$$Gap(k) := \frac{1}{B} \sum_b \log(W^*_{b,k}) - \log(W_k), \quad k = 1, \ldots, K,$$

4. **Poser** $\bar{l} := \frac{1}{B} \sum_b \log(W^*_{k,b})$.

   Calculer l'écart-type

$$sd(k) := \left[\frac{1}{B} \sum_b \left(\log\left(W^*_{k,b}\right) - \bar{l} \right)^2 \right]^{\frac{1}{2}},$$

   et **définir** $s_k := sd(k)\sqrt{1 + 1/B}$.

   Enfin, **choisir** le nombre de classes k^* comme étant l'entier positif non nul k vérifiant

$$Gap(k^*) \geq Gap(k^* + 1) - s_{k^*+1}.$$

Remarques sur l'algorithme

- Nous fixerons la constante relative au nombre maximal de classes à $K = 10$.
- Nous générerons $B = 25$ processus de Poisson durant la deuxième étape de l'algorithme.
- Il va de soi que si le jeu de données original E possède n objets, les B processus de Poisson seront également munis d'exactement n réalisations. Il s'agit ici d'une première convention à l'égard de l'intensité du processus sur la bounding box.
- La constante de rejet du modèle uniforme $\omega := \sqrt{1 + \frac{1}{B}}$ semble avoir convenu aux auteurs dans leurs simulations. Nous verrons ce qu'il en est dans nos propres simulations.
- Dorénavant, un seul algorithme partitionnera les modèles uniformes simulés : la **méthode des centres mobiles.** Cet algorithme est sélectionné pour sa simplicité en termes de programmation (voir **Annexe**). Aussi, il sera supposé que des algorithmes de classification différents partitionnent un modèle uniforme *de manière semblable.* Cette hypothèse nous permet donc d'utiliser le même algorithme de classification $\mathcal{A}'$ durant la deuxième étape, indépendamment de l'algorithme $\mathcal{A}$ utilisé dans la première étape.

– La règle d'arrêt sera interprétée de la manière suivante :

si $k^\star$ est l'entier sélectionné par la méthode, alors la quantité Gap évaluée en $k^\star + 1$ est soit plus petite que son évaluation en $k^\star$, soit plus importante mais trop peu que pour qu'il soit accepté de poursuivre le procédé.

Exemples théoriques

Le premier exemple est un jeu de données caractérisé par deux descripteurs, généré par deux vecteurs gaussiens de moyennes $\mu_1 = (0,0)$ $(n_1 = 50)$ et $\mu_2 = (5,5)$ $(n_2 = 50)$, et de matrice de dispersion $\Sigma = 0.7I$. La figure suivante nous confirme l'existence de deux classes intuitives, comme nous pouvions le supposer.

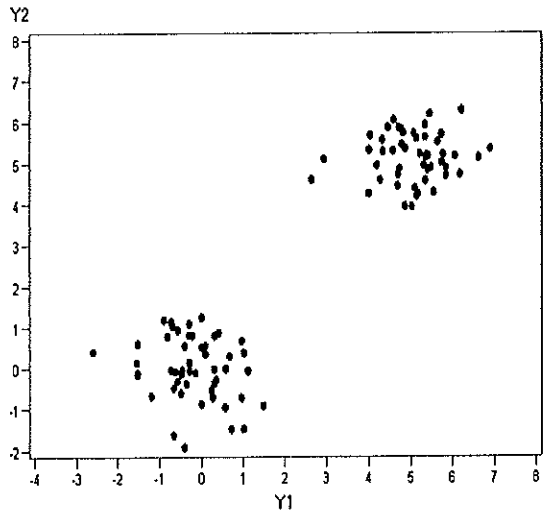


FIG. 3.3 – Jeu de données composé de deux classes naturelles, $n = 100$

Nous pouvons exécuter un algorithme de partitionnement (par exemple, la **méthode hiérarchique de Ward**, efficace sur les classes hypersphériques) en $k = 1, \ldots, 10$ classes, stocker les logarithmes des inerties intra-classes, puis réitérer l'expérience avec un processus de Poisson **moyen**, comme décrit dans l'algorithme à la page précédente. Ensuite, nous pouvons calculer les statistiques *Gap* estimées.

Remarquons que, en $k = 2$, l'allure de la courbe $\log W_k$ se démarque clairement de celle de $\log W_k^\star$. C'est en ce nombre de classes que la statistique *Gap* croît subitement (figure 3.4, page suivante). Les graphiques sont réalisés sous le logiciel MATLAB.

Le second jeu de données est un ensemble de points en deux dimensions, distribués uniformément sur le carré $[0,4]^2$, donc sans structure en classes (figure 3.5, page suivante).

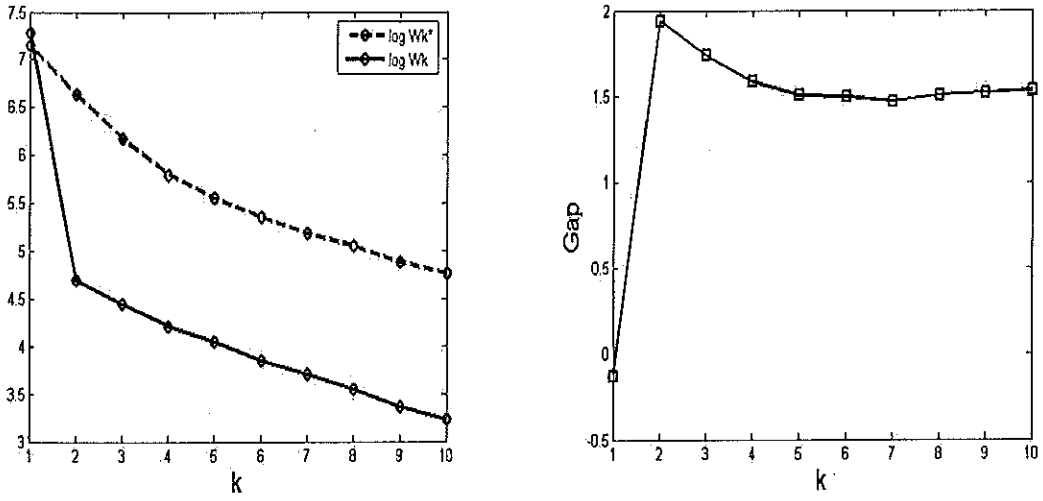


FIG. 3.4 – Valeurs des logarithmes des inerties intra-classes du jeu de données original ($\log W_k$), et du processus de Poisson ($\log W_k^\star$) (à gauche) et estimations des statistiques *Gap* (à droite), en fonction du nombre de classes

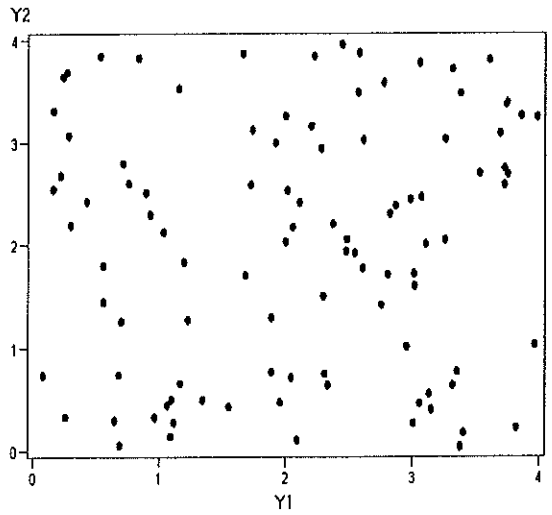


FIG. 3.5 – Jeu de données distribué uniformément, sans structure en classes, $n = 100$

L'algorithme est répété, en considérant également les logarithmes des inerties intra-classes sorties par l'algorithme de classification de Ward.

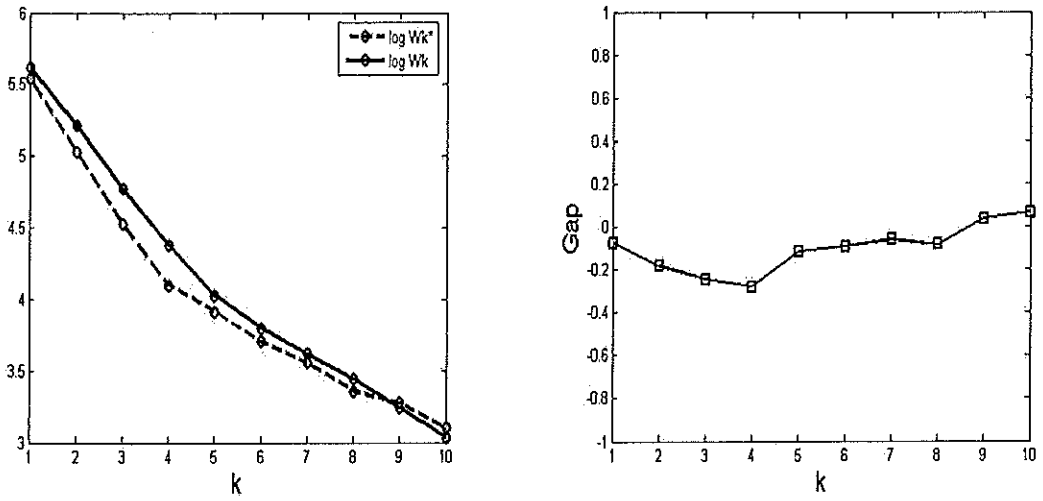


FIG. 3.6 – Valeurs des logarithmes des inerties intra-classes du jeu de données original ($\log W_k$), et du processus de Poisson ($\log W_k^\star$) (à gauche) et estimations des statistiques *Gap* (à droite), en fonction du nombre de classes

Nous pouvons remarquer sur le graphique 3.6 (à gauche) que les courbes $\log W_k$ et $\log W_k^\star$ sont presque confondues. Cette observation est logique puisque l'algorithme est ici appliqué sur **deux** processus de Poisson. Ainsi donc, aucun nombre de classes particulier ne démarque clairement les deux courbes.

Remarque

Comme vu au chapitre précédent, d'autres méthodes de détermination du nombre de classes moins récentes ont été développées. Ainsi, Gordon divise ces approches en distinguant les méthodes locales des méthodes globales. La *Gap Statistic* évalue une mesure sur tout le jeu de données, et l'optimise en fonction du nombre de classes. Il s'agit par conséquent d'une **méthode globale**. Notons que la méthode présentée rejoint l'idée de la méthode du critère de classification cubique décrite au deuxième chapitre, en ce sens qu'elle fait appel à des informations externes au jeu de données original, en l'occurrence B réalisations d'un processus de Poisson.

La plupart des méthodes globales ont le désavantage de ne pas vérifier l'absence de structure en classes. Dans notre cas, l'absence de structure en classes sera ici décidée si la règle de décision ne sélectionne aucun nombre de classes optimal sur les K tests réalisés, **ou si l'absence de structure en classes distinctes est décidée lors de la première itération de la boucle, selon la règle d'arrêt définie précédemment.**

3.4 Classes imbriquées

Les études et comparaisons réalisées au chapitre suivant démontrent empiriquement que la méthode de la *Gap Statistic* donne de bons résultats lorsqu'elle est appliquée sur des jeux de données aux classes relativement bien séparées. Posons-nous maintenant la question de savoir comment la méthode réagit sur des ensembles de classes *peu séparées.*

Dans cette section, nous présentons une courte expérience réalisée par Tibshirani [25] ayant pour but de tester la méthode sur des données non séparées. Nous considérerons une série de jeux de données, constitués d'exactement $n = 100$ objets, construits par la génération de deux sous-populations bivariées normales ($n_1 = n_2 = 50$), de moyennes respectives $\mu_1 = (0, 0)$ et $\mu_2 = (\Delta, 0)$ et de dispersion commune $\Sigma = I$.

Pour chaque jeu de données, l'auteur a exécuté la méthode un certain nombre de fois, et enregistré la proportion d'affirmations d'absence de structure en classes ($k^\star = 1$). Il a également enregistré la proportion d'**objets mal positionnés** (*overlapping points*) du jeu de données en question. Ces objets sont tels qu'ils appartiennent à une classe générée mais placés dans l'enveloppe convexe de l'autre classe. L'expérience a été réitérée pour 10 valeurs de $\Delta \in [0, 5]$. Les résultats sont repris par la figure 3.7.

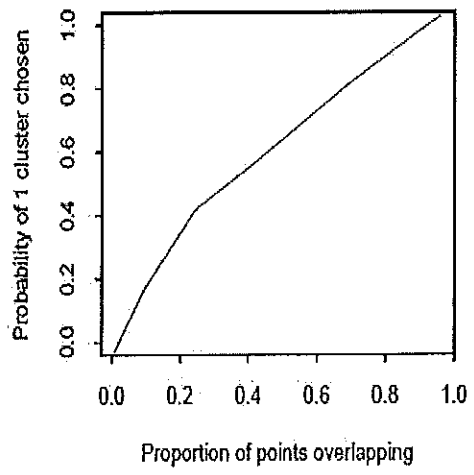


FIG. 3.7 – Proportion de "$k^\star = 1$" rendus, fonction de la proportion d'objets mal placés [25]

Ainsi, si la proportion d'objets mal placés est p, alors la probabilité d'avoir pour output $k^\star = 1$ est également p.

3.5 Estimation du nombre de composantes principales

Considérons maintenant le problème qu'est l'estimation de composantes linéaires principales nécessaires à l'approximation optimale d'un nuages de points. Comme précédemment, nous avons comme données x_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, p$.

Soit W_k' **l'erreur de reconstruction** obtenue en utilisant k composantes principales pour approximer les données. Précisément, soit $\hat{x}_{ij}^k$ la projection de l'objet i sur la k^{eme} composante principale approximante. Alors

$$W_k' = \sum_i \sum_j (x_{ij} - \hat{x}_{ij}^k)^2.$$

Dans le but d'approximer le bon nombre de composantes principales $k^\star$, il suffira d'appliquer la méthode de la *Gap Statistic* aux quantités W_k' et $W_k^{\star\,\prime}$ à la place de W_k et $W_k^\star$, où $W_k^{\star\,\prime}$ est l'erreur de reconstruction obtenue en utilisant la même A. C. P. sur le processus de Poisson généré dans la bounding box du jeu de données original.

Tibshirani [25] a réalisé l'expérience sur un jeu de données reprenant 100 objets, simulés selon le modèle

$$\begin{aligned} Y_{1,2} &\sim U([0,1]), \\ Y_{3,4,5} &\sim N(0, 0.25), \end{aligned}$$

de sorte qu'il soit possible de représenter le jeu de données en deux dimensions, issues des deux principales distributions simulées. Les résultats de la méthode de la *Gap Statistic* appliquée à ce concept sont repris par les deux figures suivantes :

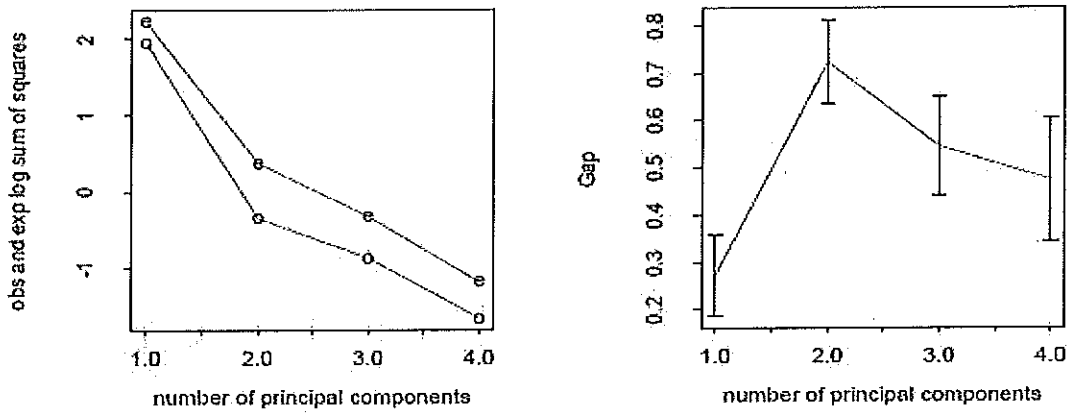


FIG. 3.8 – Application de la méthode à l'estimation du nombre de composantes principales [25]

C'est en $k = 2$ que la quantité *Gap* trouve son maximum. L'output correspond donc bien à nos attentes : $k^\star = 2$.

Cette application est ici expliquée par l'exemple. Son développement complet nécessite des études complémentaires.

3.6 Application de la méthode à un jeu de données réel

Dans leur article [25], les auteurs se sont intéressés à un exemple réel médical. Il s'agit d'une matrice réelle de dimension 64 × 1000 (64 tumeurs **diagnostiquées**, décrites par 1000 gènes). L'expertise n'a naturellement pas été considérée dans la classification utilisée, en l'occurrence celle du lien moyen. Le dendrogramme obtenu est le suivant.

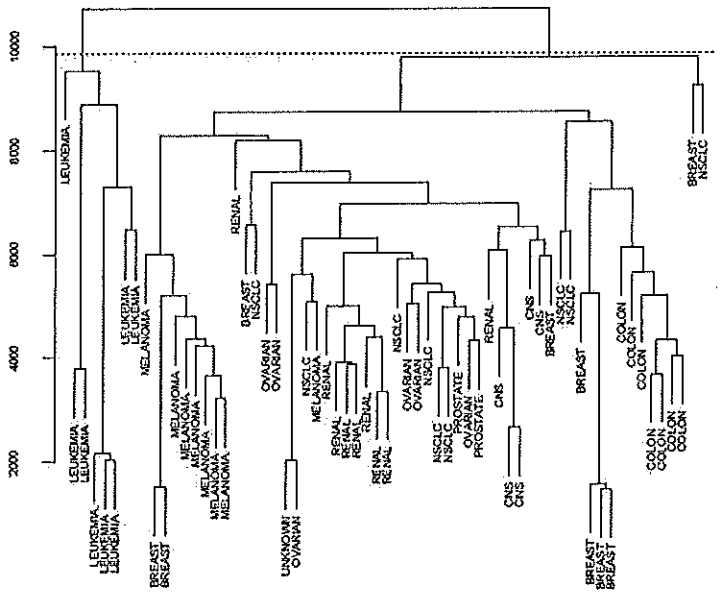


FIG. 3.9 – Dendrogramme obtenu sur le jeu de données médical par la méthode du lien moyen [25]

Il n'est pas surprenant de constater que les tumeurs du même type sont groupées.

Voyons maintenant les résultats sortis par la méthode de la *Gap Statistic* (figure 3.10).

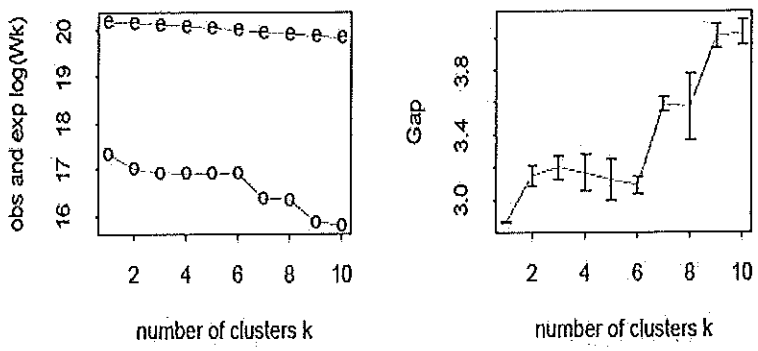


FIG. 3.10 – Résultats de la méthode sur le dendrogramme [25]

La méthode sort donc $k^* = 2$. Cette coupure correspond au trait discontinu sur le dendrogramme. La méthode sépare logiquement les leucémies des autres cancers.

Chapitre 4

Applications

Ce chapitre concerne maintenant l'étude de l'efficacité de la méthode de la Gap Statistic sur des jeux de données plus ou moins théoriques. Pour réaliser ces tests, il a fallu naturellement programmer la méthode. Le choix s'est porté sur le langage C. Un package de fichiers est attaché à ce mémoire. Il contient le code du programme, et l'exécutable associé. Ce programme est commenté. Avant de poursuivre dans ce chapitre, il est recommandé au lecteur de **lire attentivement l'Annexe A**. Cette annexe constitue le mode d'emploi du programme et contient également le code source en question. Quant aux résultats de la méthode, ils seront comparés aux résultats fournis par les méthodes de Calinski et Harabasz, de Duda et Hart, et du C. C. C., expliquées au deuxième chapitre. Afin de ne pas alourdir les outputs de ces méthodes, nous ne leur associerons que les k^* qu'ils fournissent. Les noms des méthodes de classification hiérarchique seront mis en gras à partir du moment où ils partitionnent le jeu de données de manière intuitive. Sur certaines données, les valeurs de la statistique *Gap* seront tracées. De tels graphiques permettent une interprétation claire de la règle d'arrêt énoncée au troisième chapitre. Pour chaque jeu de données, un commentaire final est réalisé.

Les symboles "OK" signifient que la méthode de classification fournit une partition intuitive du jeu de données exposé.

4.1 Application à deux jeux de données composés de classes hypersphériques

Nous nous plaçons en premier lieu dans le cadre très confortable des jeux de données composés de classes hypersphériques. Ce type de classes est le plus susceptible d'être rencontré dans la réalité.

Un jeu de données théorique aux classes hypersphériques

Voyons le jeu de données théorique aux classes hypersphériques suivant, avec $p = 2$, $n = 150$ (figure 4.1). Il est clair que la partition intuitive de cet ensemble doit contenir trois classes.

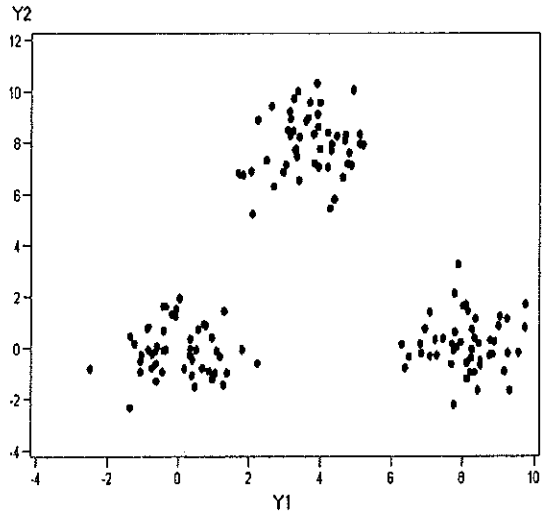


FIG. 4.1 – Jeu de données théorique composé de classes hypersphériques

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK	OK	OK	OK	OK
C-H	3	3	3	3	3
D-H	3	3	3	3	3
CCC	3	3	3	3	3
Gap S.	3	3	3	3	3

Sur ce jeu de données, la méthode est parfaite. Elle n'en tire cependant aucun mérite, étant donné la structure du jeu, et ne se démarque pas des autres méthodes. Pour $k = 3$, chacune des cinq méthodes de classification utilisées a fourni une partition formée des 3 classes intuitives.

Les données de Ruspini

Voyons le jeu de données de Ruspini, célèbre dans la littérature [24], avec $p = 2$, $n = 75$ (figure 4.2). Cet ensemble de points devrait intuitivement être partitionné en quatre classes.

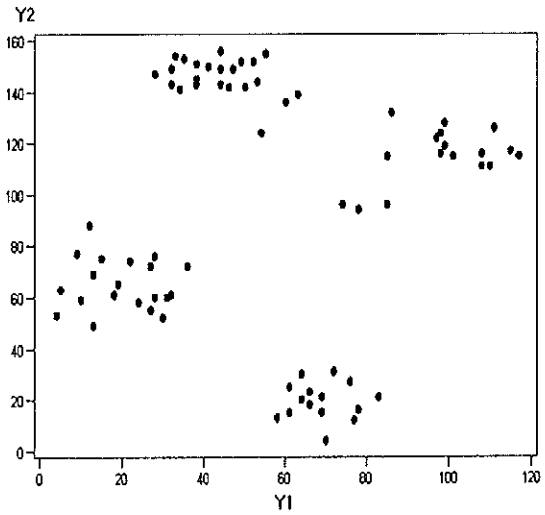


FIG. 4.2 – Jeu de données de Ruspini

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK	OK		OK	OK
C-H	4	4	4	4	4
D-H	3	3	3	3	4
CCC	4	4	4	4	4
Gap S.	4	4	4	4	4

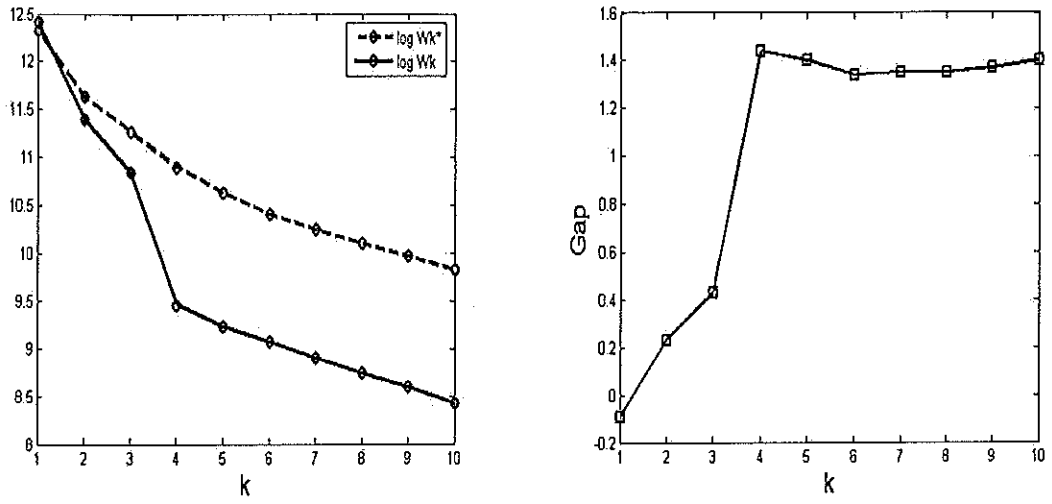


FIG. 4.3 – Output du programme pour les données de Ruspini, sur base d'un partitionnement du jeu de données par la méthode de Ward

Sur les données de Ruspini, la méthode est également efficace (si du moins, nous considérons que ces données comportent bel et bien quatre classes distinctes). Seule la méthode de Duda & Hart semble hésiter quant au nombre de classes, ce qui n'est pas spécialement étonnant vu la structure du jeu de données. En effet, nous pouvons souligner la proximité des deux groupes d'objets pour lesquels la variable Y_2 est la plus élevée. Quant aux partitions en quatre classes, elles sont intuitives pour chacune des méthodes de classification, excepté celle du lien maximum. La figure 4.3 nous informe du fait que la statistique *Gap* est clairement croissante de $k = 1$ à $k = 4$. La valeur $Gap(4)$ ($\simeq 1.45$) est cependant plus élevée que $Gap(5) \simeq 1.41$. L'assertion $k^* = 4$ est donc privilégiée.

Les valeurs des statistiques $Gap(k)$ ne sont pas complètement reprises ici, afin de ne pas alourdir le texte. Le lecteur peut cependant, moyennant un input adapté comme expliqué en **Annexe**, exécuter le programme afin de les obtenir.

4.2 Application à deux jeux de données sans structure en classes

Nous nous intéressons maintenant à deux jeux de données, générés selon deux distributions de modèles nuls : le modèle uniforme et le modèle normal.

Un jeu de données uniforme

Voyons une réalisation d'un processus de Poisson homogène, généré sur l'hypercube $[0, 5]^4$, avec $p = 4$, $n = 100$.

La dimension du jeu étant supérieure à 2, une **analyse en composantes principales** s'avère nécessaire afin que le jeu de données soit projeté en deux dimensions (figure 4.4), dans un but de représentation en deux dimensions. Quant aux partitionnements du jeu

de données sur lesquels sont basées les méthodes de détermination du nombre de classes, elles sont réalisées **à partir du jeu de données original** et non pas sur base de l'A. C. P..

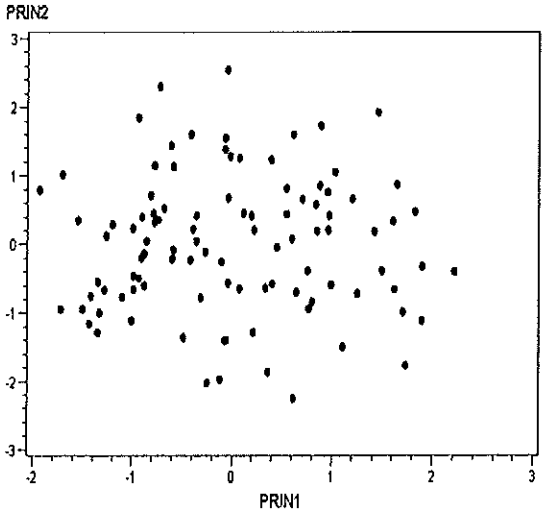


FIG. 4.4 – A. C. P. sur le jeu de données uniforme

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	X	X	X	X	X
C-H	7	4	2	4	2
D-H	7	3	2	4	2
CCC	1	1	1	1	1
Gap S.	1	1	1	1	1

La méthode sort les mêmes outputs que la méthode CCC. Quant aux deux premières, elles sortent des résultats intraitables, puisqu'incapables de vérifier l'absence de structure en classes. Les symboles "X" sont inscrits ici pour rappeler qu'on ne peut logiquement pas affirmer si une méthode partitionne un jeu de données dépourvu de structure en classes de manière intuitive ou non.

Un jeu de données normal

Voyons une réalisation d'une distribution, générée autour du vecteur nul de dimension $p = 4$, de disperson $\Sigma = I$, $n = 100$.

La dimension du jeu étant supérieure à 2, une **analyse en composantes principales** s'avère nécessaire afin que le jeu de données soit visionnable en deux dimensions (figure 4.5). Les procédés de classification sont naturellement réalisés sur le jeu de données original, décrit par les $p = 4$ variables descriptives.

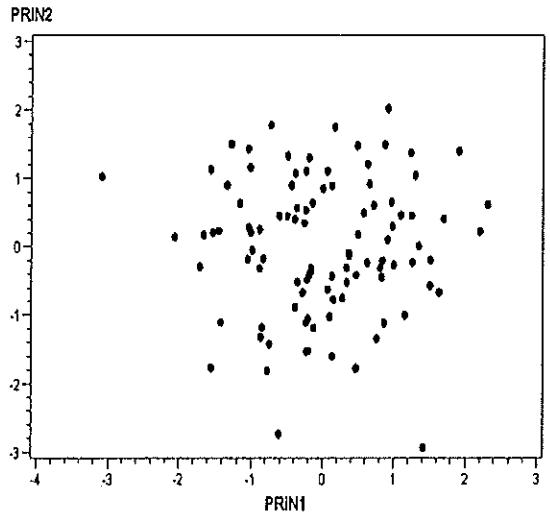


FIG. 4.5 – A. C. P. sur le jeu de données normal

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	X	X	X	X	X
C-H	2	5	4	6	3
D-H	2	5	3	6	3
CCC	1	1	1	1	1
Gap S.	1	1	1	1	1

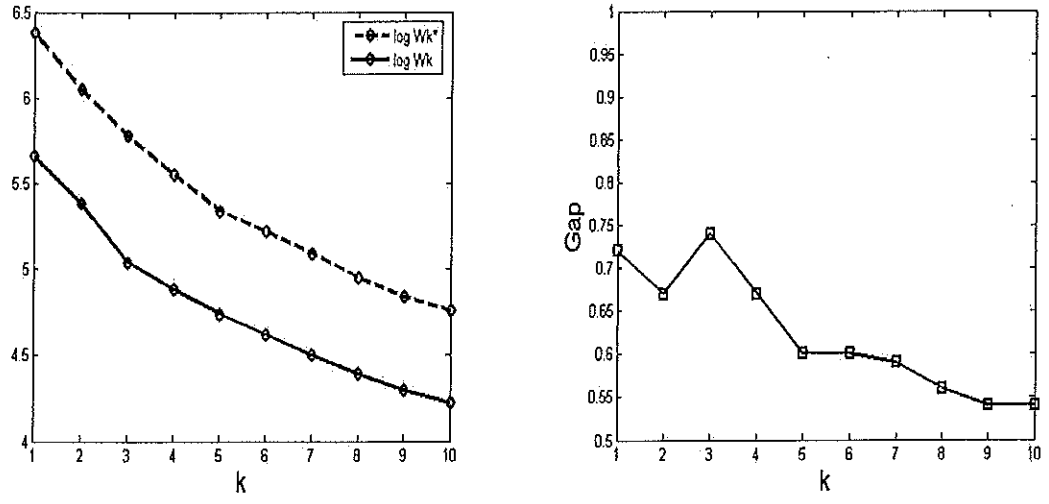


FIG. 4.6 – Output du programme pour la réalisation normale, sur base d'un partitionnement de cette réalisation par la méthode de Ward

La conclusion pour ce cas est similaire à la précédente. Remarquons de plus que la courbe du logarithme de l'inertie intra-classe de la partition du jeu de données normal

est bien en-dessous, pour tout k, de celle associée au processus de Poisson homogène. En effet, un jeu de données généré selon une distribution normale est constitué d'une majorité d'individus *proches* du centroïde de la classe. Ainsi, toute classe d'un *jeu normal* a une inertie moins importante que toute classe **de même taille** d'un *jeu uniforme*, à partir du moment où les deux jeux sont générés **dans la même bounding box.**

4.3 Application à deux jeux de données aux structures allongées

Nous nous intéressons maintenant à deux jeux de données aux structures allongées.

Un jeu de données allongé aux deux variables dépendantes

Examinons le jeu de données aux classes hyperellipsoïdales suivant, avec $p = 2$, $n = 400$ (figure 4.7).

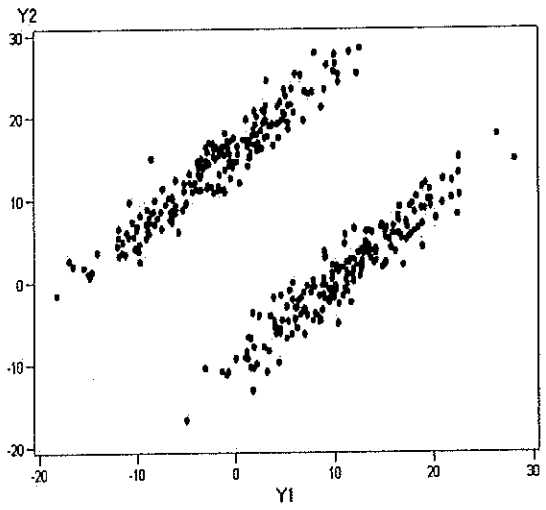


FIG. 4.7 – Un premier jeu de données composé de classes allongées

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK	OK			OK
C-H	2	2	6	3	2
D-H	2	2	4	3	2
CCC	2	2	1	1	2
Gap S.	2	2	1	1	2

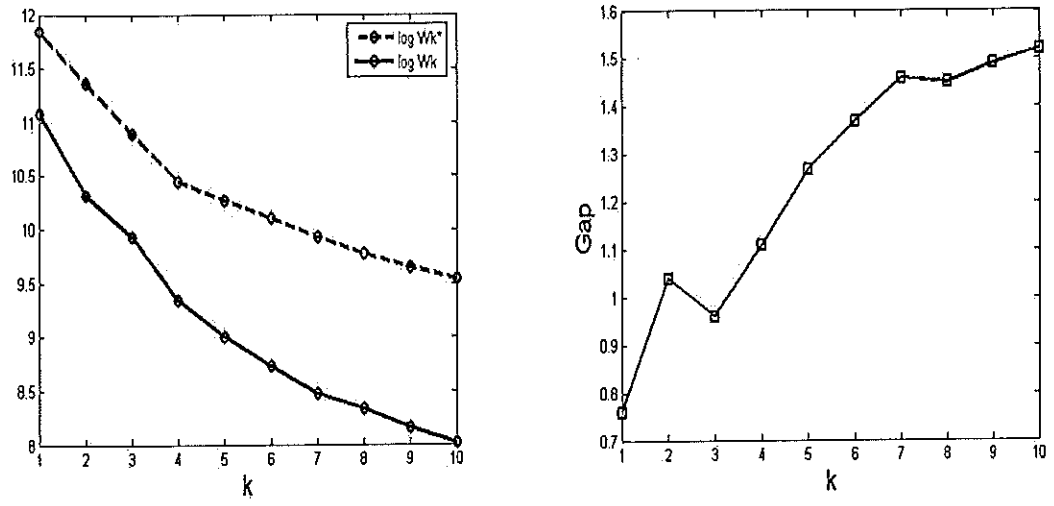


FIG. 4.8 – Output du programme pour le jeu de données composée de deux classes hyperellipsoïdales, sur base d'un partitionnement de ce jeu de données par la méthode de Ward

La méthode fournit les mêmes outputs que la méthode CCC. Quant aux deux premières, elles sortent des résultats tantôt étonnants, tantôt intuitivement justes, en fonction de l'efficacité de l'algorithme de classification sur le jeu de données en question. L'output $k = 2$ est rendu par toutes les méthodes de détermination du nombre de classes, lorsqu'elles sont exécutées sur des partitions en deux classes intuitives, c'est-à-dire composées des deux classes allongées. Sur la figure 4.8, on observe que la valeur $Gap(1)$ est plus petite que la valeur $Gap(2)$. On observe également une décroissance nette de la statistique $Gap(k)$ de $k = 2$ à $k = 3$, d'où le rejet du modèle nul en $k = 2$.

Un jeu de données allongé par rapport à une seule variable

Considérons le jeu de données suivant, généré de telle sorte que trois classes allongées intuitives se distinguent. Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK				OK
C-H	3	3	3	3	6
D-H	2	3	3	3	2
CCC	3	1	1	1	6
Gap S.	3	1	1	1	3

Pour ce jeu de données, l'output de la méthode n'est pas tout à fait similaire à celui du CCC. Il s'agit du premier point de divergence entre les deux méthodes. Notons que la méthode est la seule à sortir deux résultats intuitifs sur base de la partition intuitive en $k = 3$ classes, rendue par les méthodes du lien minimum et de Ward. Sur la figure 4.10, la valeur de la fonction Gap est strictement croissante de $k = 1$ à $k = 3$ et $Gap(4)$ est plus petite que $Gap(3)$. D'où $k^* = 3$.

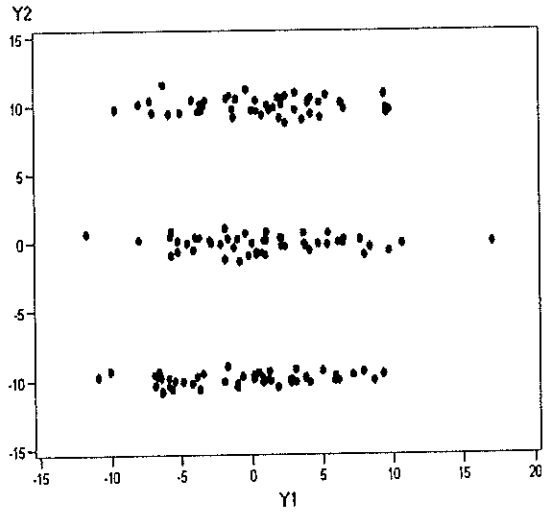


FIG. 4.9 – Un second jeu de données aux classes hyperellipsoïdales

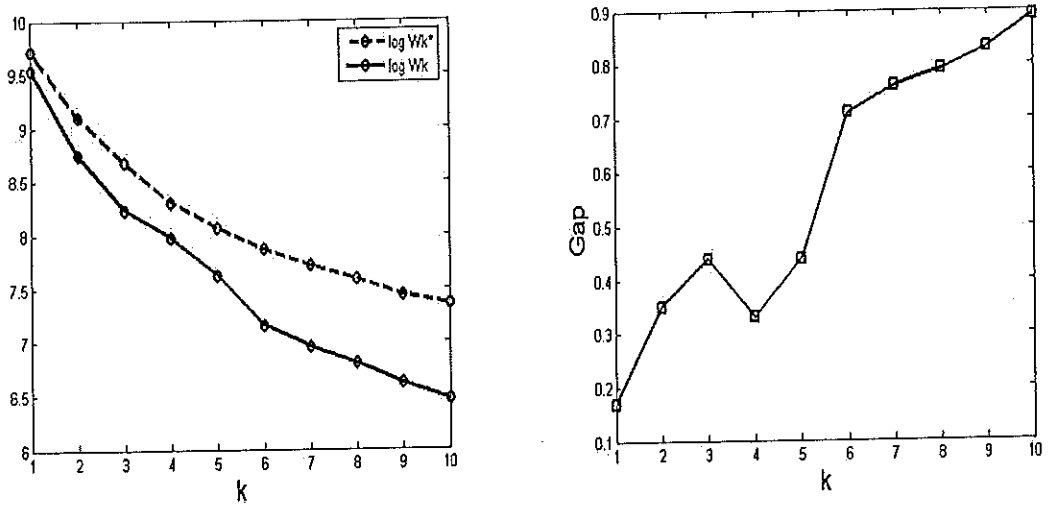


FIG. 4.10 – Output du programme pour le jeu de données divisé en trois classes allongées, sur base d'un partitionnement de ce jeu de données par la méthode de Ward

4.4 Application à deux jeux de données issus de la réalité

Nous nous intéressons maintenant à deux jeux de données, caractérisés par des variables descriptives évaluées sur des échantillons réels.

Les iris de Fisher

Contrairement aux ensembles de données traités jusqu'à présent, ces données sont réelles. Elles se composent de 150 fleurs provenant de trois variétés d'iris : *setosa*, *versicolor*, *virginica* [2]. Sur chacune des plantes, R. A. Fisher a observé quatre descripteurs. Il s'agit de la longueur et de la largeur des pétales et des sépales des iris. Chaque variété est représentée par 50 fleurs.

La dimension du jeu étant supérieure à 2, une **analyse en composantes principales** s'avère nécessaire afin que le jeu de données soit visionnable en deux dimensions.

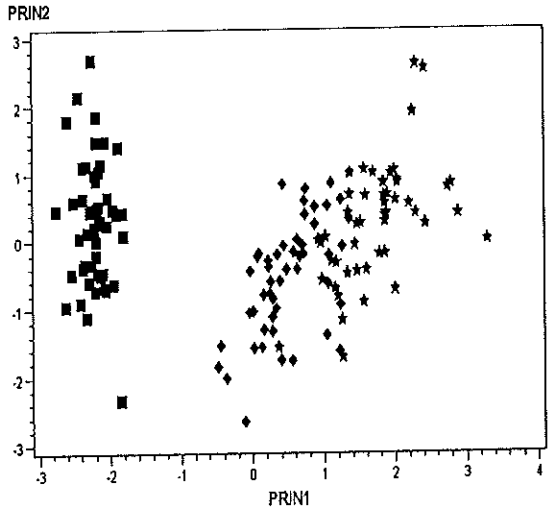


FIG. 4.11 – A. C. P. réalisée sur le jeu de données de Fisher

Remarquons que, malgré le fait que le jeu de données soit constitué de trois classes réelles, le nombre intuitivement decelé sur l'A. C. P. est clairement de deux. En effet, il est difficile de distinguer les classes caractérisées par les **losanges** et les **étoiles**. Cependant, une représentation A. C. P. efface parfois certaines caractéristiques de la structure propre au jeu de données.

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
				$\simeq$ OK	
C-H	2	2	4	3	3
D-H	2	2	3	2	2
CCC	2	2	1	3	3
Gap S.	2	4	4	3	6

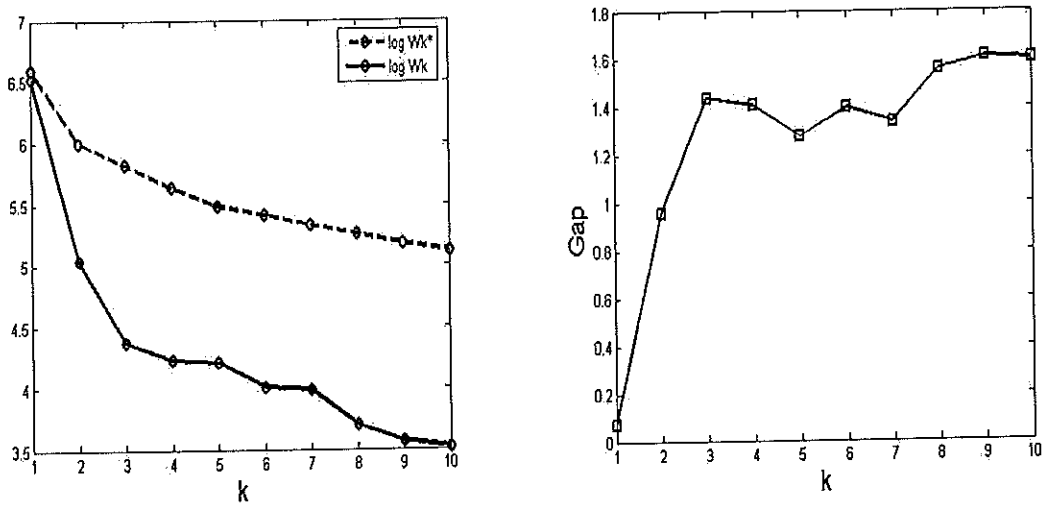


FIG. 4.12 – Output du programme pour les iris de Fisher, sur base d'un partitionnement de ce jeu de données par la méthode des centroïdes

C'est uniquement sur base de la partition rendue par l'algorithme des centroïdes que la méthode opte pour l'assertion $k^* = 3$. La méthode des centroïdes ne partitionne pas, pour $k = 3$, le jeu de données comme indiqué par l'expertise. Il s'agit tout de même la méthode dont la partition (figure 4.13) *ressemble* le plus à la partition rendue par Fisher, en comparaison avec les quatre autres. En effet, la classe représentée par des **carrés** est maintenue telle quelle. Quant aux deux autres classes, la méthode des centroïdes est le seul algorithme capable de plus ou moins bien les reproduire. Aussi, l'hyperplan qui les sépare sur l'expertise est approximativement similaire à l'hyperplan qui les sépare sur la partition des centroïdes.

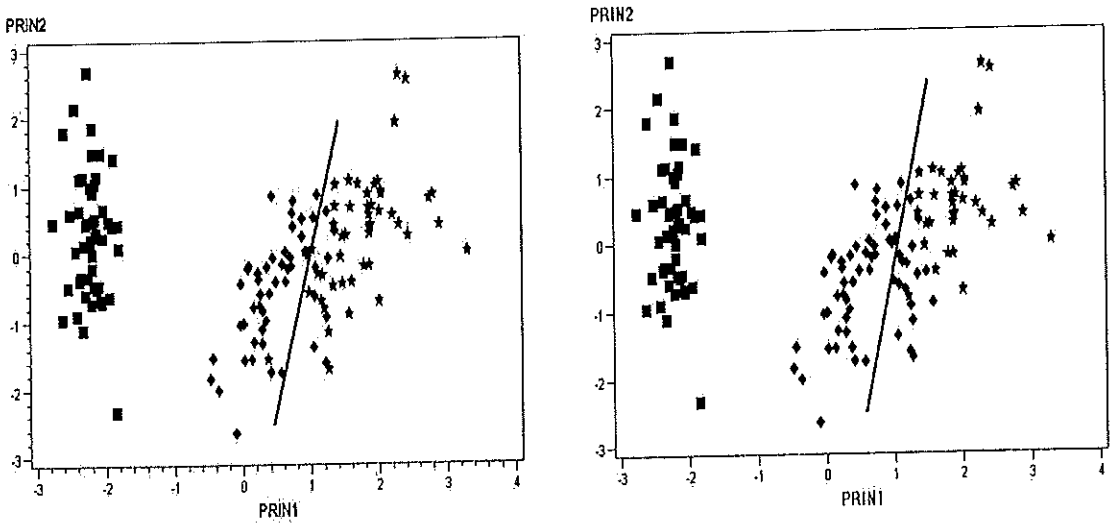


FIG. 4.13 – Partition du jeu de données de Fisher par expertise (à gauche), par la méthode des centroïdes (à droite), avec hyperplans séparant les classes "étoiles" et "losanges"

Des données européennes

Il s'agit d'un jeu de données, composé de 26 pays situés sur le sol européen [27]. L'étude date de 1979, et a pour but d'analyser les répartitions de main d'oeuvre dans différents secteurs industriels. Les 9 variables descriptives sont apparentées aux labels suivants :

- **Agr** : pourcentage de main d'oeuvre employée dans l'agriculture,
- **Min** : pourcentage employé dans les mines,
- **Man** : pourcentage employé dans l'industrie de fabrication,
- **PS** : pourcentage employé dans les industries fournissant de l'énergie,
- **Con** : pourcentage employé dans la construction,
- **SI** : pourcentage employé dans les services,
- **Fin** : pourcentage employé dans les finances,
- **SPS** : pourcentage employé dans les services personnels et sociaux,
- **TC** : pourcentage employé dans les transports et communications.

La dimension du jeu étant supérieure à 2, une **analyse en composantes principales** s'avère nécessaire afin que le jeu de données soit visionnable.

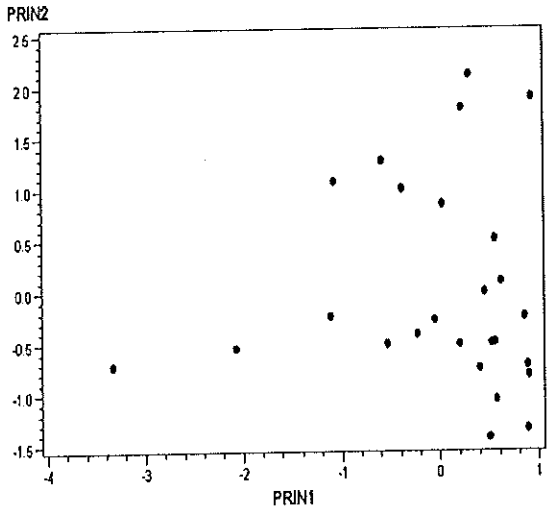


FIG. 4.14 – A. C. P. réalisée sur le jeu de données européen

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	X	X	X	X	X
C-H	3	2	3	2	3
D-H	3	2	2	2	2
CCC	1	1	1	1	1
Gap S.	1	1	1	1	1

Une étude sur ce jeu de données [12], externe à ce mémoire, a démontré que les algorithmes de classification hiérarchiques ont tendance, pour $k = 2$, à écarter les pays que sont la Turquie, la Yougoslavie et la Grèce (pour lesquels PRIN1 est petit), en raison de la main-d'oeuvre importante employée dans leurs secteurs agricoles.

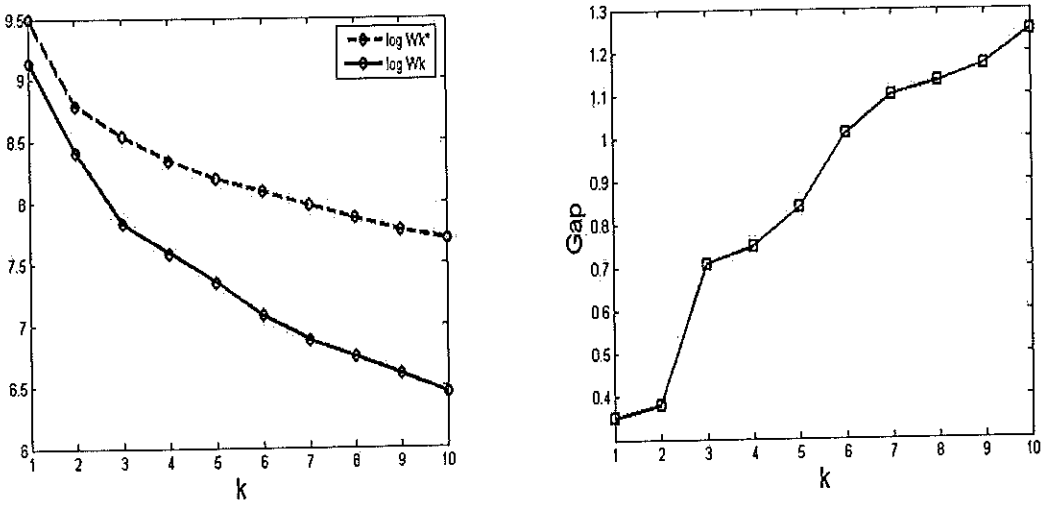


FIG. 4.15 – Output du programme pour les données européennes, sur base d'un partitionnement de ce jeu de données par la méthode de Ward

Nous remarquerons cependant, qu'au vu de l'A. C. P. (figure 4.14), nous pourrions déduire l'absence de structure en classes du jeu de données. Cette hypothèse est conservée par la méthode. La méthode du critère de classification cubique fournit également, sur base de chaque algorithme de classification, une décision d'absence de structure en classes. Quant à la figure 4.15, elle démontre toute l'importance de la constante de rejet ω citée dans l'algorithme général développé au troisième chapitre. En effet, il est clair que

$$Gap(1) < Gap(2)$$

mais la valeur $Gap(2)$ n'est pas assez grande par rapport à $Gap(1)$ pour que la méthode ne privilégie un nombre de classes supérieur à 1. Une telle assertion aurait été considérée si, comme indiqué au troisième chapitre :

$$\begin{array}{lcl} Gap(1) & < & Gap(2) - \omega sd(2) \\ & \Longleftrightarrow & \\ 0.35 & < & 0.38 - 0.099, \end{array}$$

avec $sd(2)$ comme écart-type des logarithmes des inerties intra-classes du processus de Poisson partitionné en deux classes.
La dernière assertion étant fausse, c'est en $k = 2$ que la règle d'arrêt rejette le modèle nul.

4.5 Application à d'autres jeux de données

Nous appliquons ici la méthode à divers jeux de données ayant tous une particularité bien précise.

4.5.1 Deux petites classes bien séparées

Afin de tester la méthode sur des petits jeux de données (pour n petit), nous nous intéresserons au jeu de données suivant, avec $p = 2$, $n = 30$. Deux classes y sont nettement distinctes.

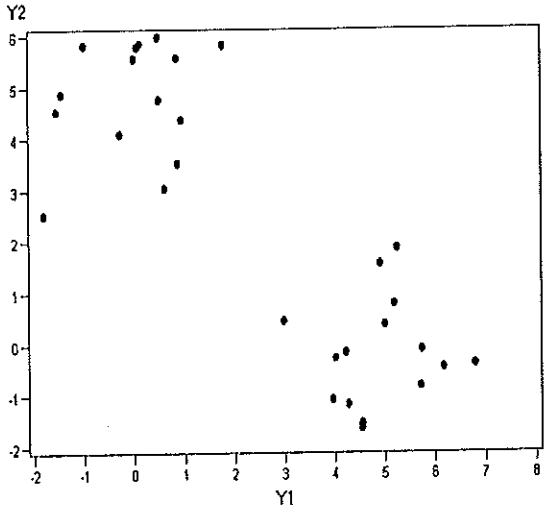


FIG. 4.16 – Jeu de données divisé en deux classes de petite taille

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK	OK	OK	OK	OK
C-H	2	2	2	2	2
D-H	2	2	2	2	2
CCC	2	2	2	2	2
Gap S.	2	2	2	2	2

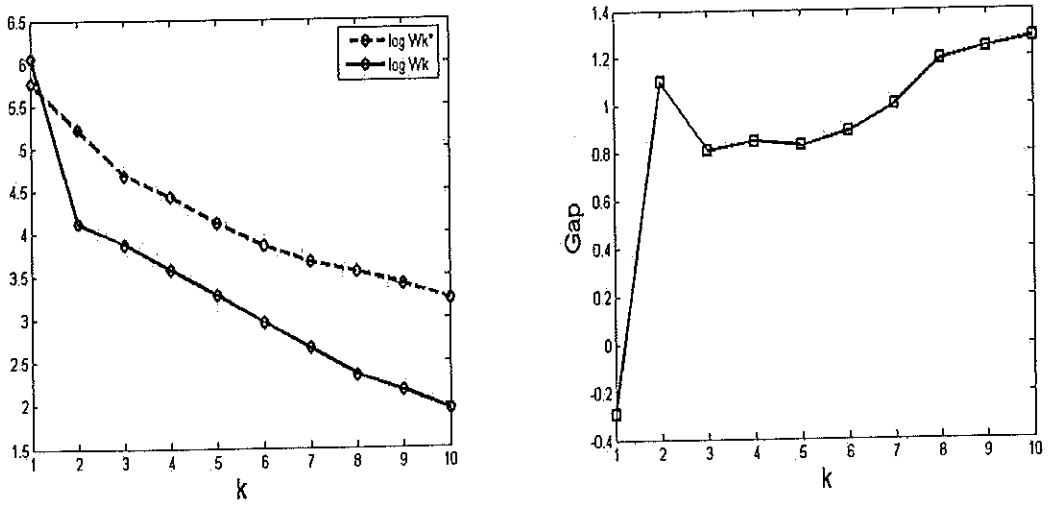


FIG. 4.17 – Output du programme pour le jeu de données, sur base d'un partitionnement par la méthode de Ward

La méthode semble bien réagir aux jeux de données constitués de petites classes. En effet, la figure 4.17 révèle un pic net de la fonction *Gap* en $k = 2$.

4.5.2 Un jeu de données grand-petit

Soit le jeu de données suivant (figure 4.18), composé d'une petite classe C_1 ($n_1 = 10$) et d'une classe C_2 plus importante ($n_2 = 100$). Ces deux classes ont été composées selon deux modèles normaux centrés respectivement en (0,0) et (5,5). L'analyse d'un tel jeu de données a pour but de vérifier si les tailles des classes intuitives perturbent ou non la méthode.

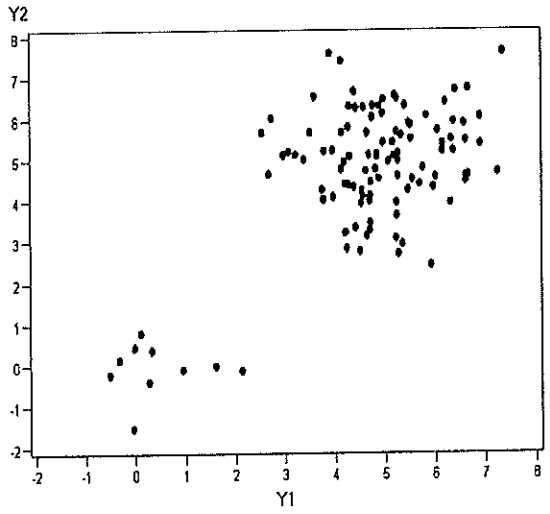


FIG. 4.18 – Jeu de données composé de deux classes de tailles différentes

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK	OK	OK	OK	OK
C-H	2	2	2	2	2
D-H	2	2	2	2	2
CCC	2	2	2	2	2
Gap S.	2	2	2	2	2

Le caractère *grand-petit* n'affecte aucune des méthodes présentées au deuxième chapitre. Elle n'affecte pas non plus la méthode qui constitue le sujet de ce mémoire. Quant à la figure 4.19, elle est relativement semblable à la figure 4.17. En effet, les deux courbes *Gap*, sur chacun des graphiques, sont quasiment identiques et présentent le même premier maximum local en $k = 2$.

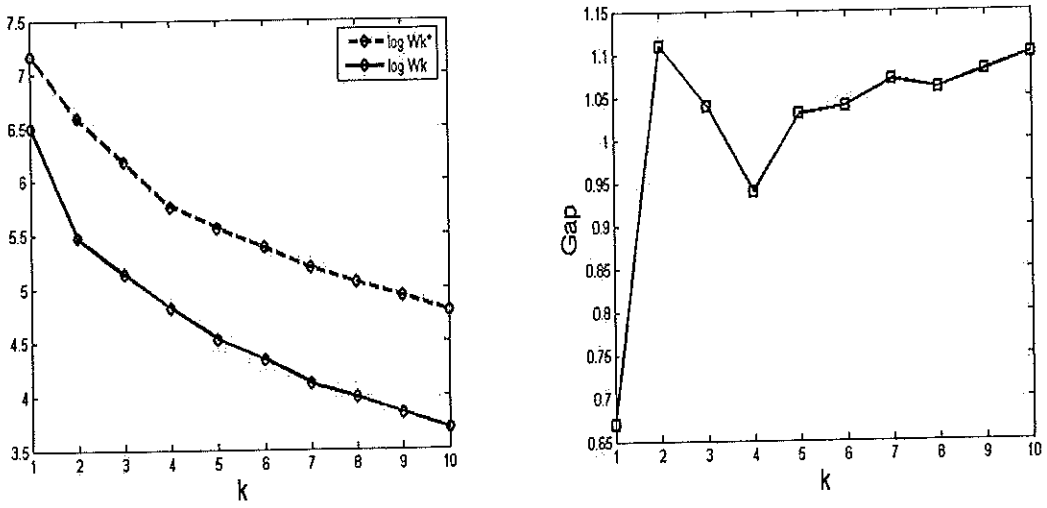


FIG. 4.19 – Output du programme pour le jeu de données, sur base d'un partitionnement par la méthode de Ward

4.5.3 Deux classes reliées par un pont

Soit le jeu de données suivant, composé de deux classes intuitivement distinctes reliées par un pont, avec $n = 165$ et $p = 2$.

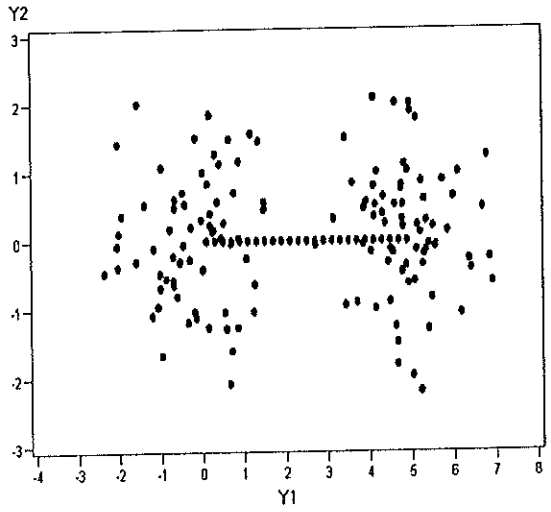


FIG. 4.20 – Deux classes reliées par un pont

Le tableau d'outputs est le suivant :

		OK	OK	$\simeq$ OK	OK
	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
C-H	2	2	2	2	2
D-H	3	2	2	2	2
CCC	1	2	2	2	2
Gap S.	1	2	2	2	2

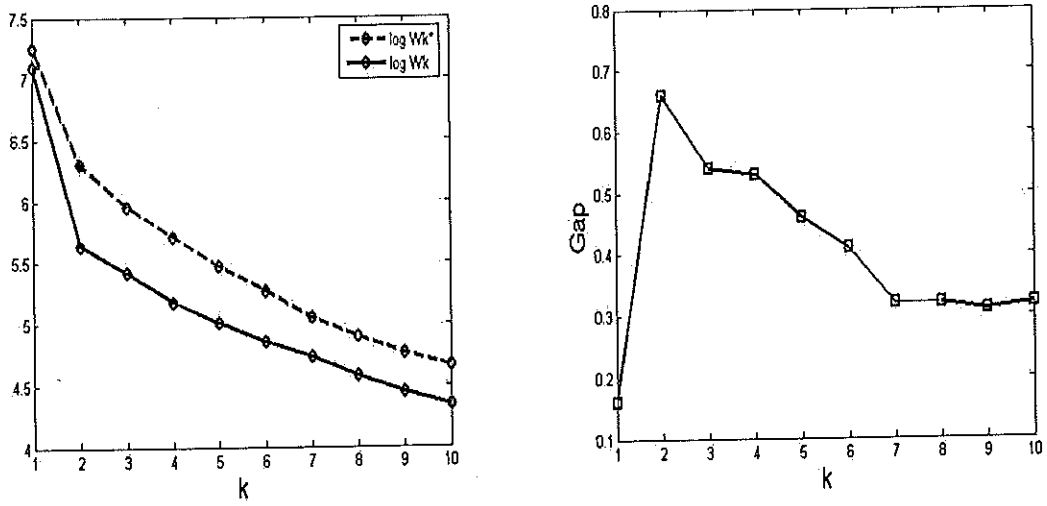


FIG. 4.21 – Output du programme pour le jeu de données, sur base d'un partitionnement par la méthode de Ward

La méthode est efficace sur toutes les partitions de base, excepté celle rendue par la méthode du lien minimum. La figure 4.21 montre d'ailleurs un pic important de la fonction *Gap* en $k = 2$. Rien de vraiment étonnant, étant donné que la partition fournie par le lien minimum, pour $k = 2$, est loin d'être intuitive, comme le montre la figure suivante 4.22. L'ajout d'un pont reliant deux classes intuitivement distinctes a pour but de *saboter* les partitions rendues par la méthode du lien minimum. C'est l'**effet de chaînage.** Cet exemple de jeu de données nous rappelle à quel point le résultat d'une méthode de détermination du nombre de classes dépend du type de partitionnement sur lequel elle est construite.

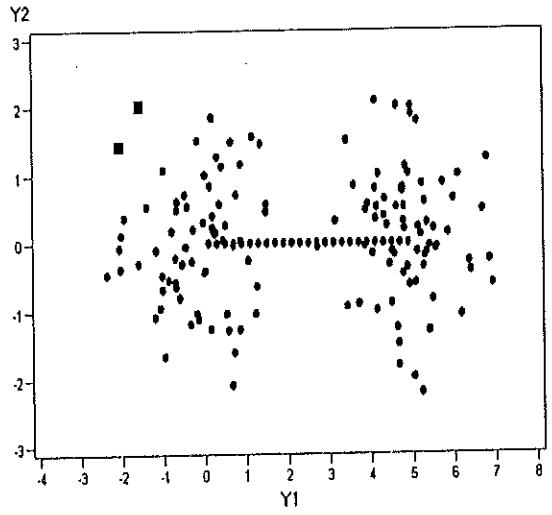


FIG. 4.22 – Partition rendue par la méthode du lien minimum

4.5.4 Un jeu de données couronne-centre

Le jeu de données suivant est très théorique, puisque peu plausible empiriquement (figure 4.23). Remarquons que les deux classes intuitives (la couronne et son centre) possèdent approximativement le même centroïde.

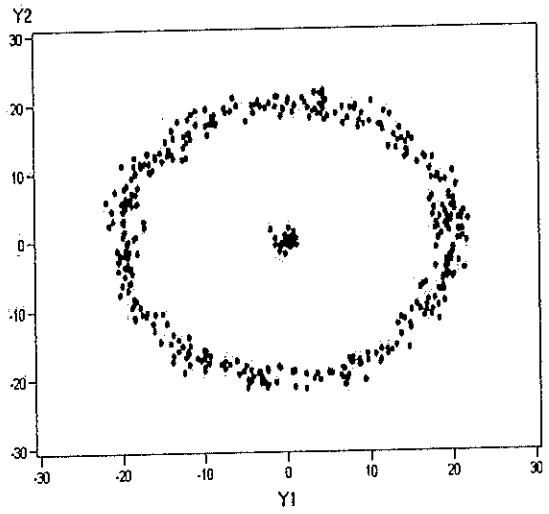


FIG. 4.23 – Jeu de données constitué d'une couronne et de son centre

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK				
C-H	7	4	6	4	3
D-H	7	4	4	3	3
CCC	1	1	1	1	1
Gap S.	1	1	1	1	1

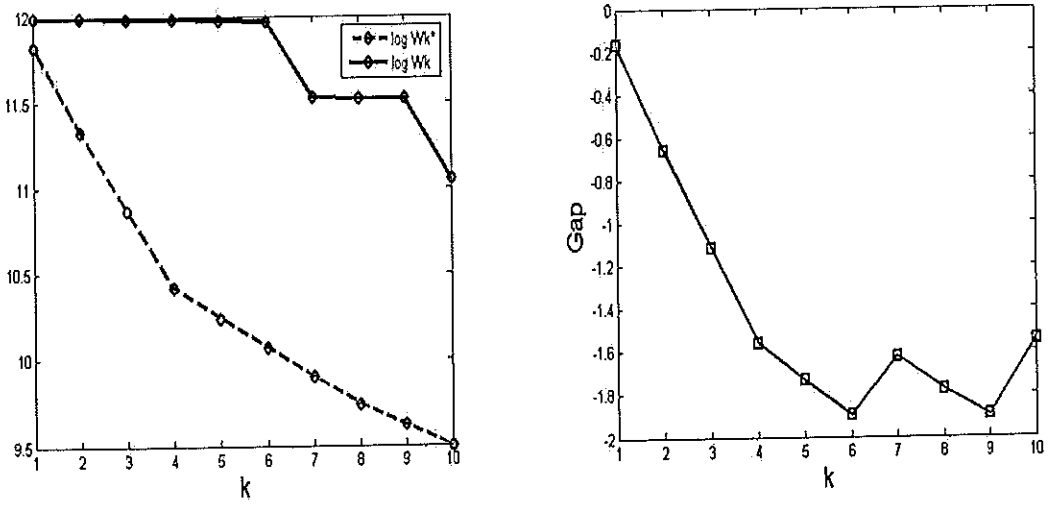


FIG. 4.24 – Output du programme pour le jeu de données, sur base d'un partitionnement par la méthode du lien minimum

Sur ce jeu de données, uniquement la méthode de classification du lien minimum rend la partition intuitive. A partir de cette classification, la méthode étudiée et la méthode du CCC ont pour output commun $k^\star = 1$. Remarquons également que, sur la figure 4.24, les valeurs $\log W_1$ et $\log W_2$ sont presque égales. Ceci est tout à fait normal : la classe C_{co}, constituée de la couronne, et la classe C_{ce}, constituée du centre, ont presque le même centroïde. Il est donc normal que la quantité $Gap(1)$ soit plus élevée que la quantité $Gap(2)$, d'où la décision prise par la règle d'arrêt de l'algorithme défini au chapitre précédent.

4.5.5 Un jeu de données aux classes non séparables par un hyperplan

Le jeu de données suivant est formé de 3 classes intuitives (figure 4.25).

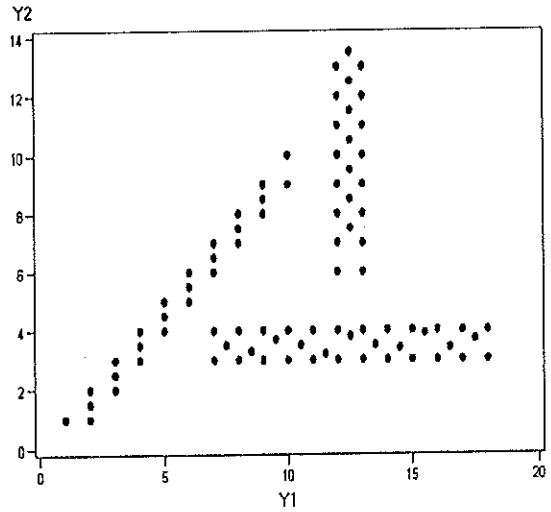


FIG. 4.25 – Un jeu de données aux classes non séparables par un hyperplan

Ces classes constituent approximativement les côtés d'un triangle. Les taille et dimension du jeu de données sont respectivement de 85 et 2. La particularité de ce jeu de données est que les classes qui le constituent ne sont pas séparables par un hyperplan.

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK				
C-H	3	4	4	4	4
D-H	3	3	3	2	3
CCC	1	1	1	1	1
Gap S.	1	1	1	1	1

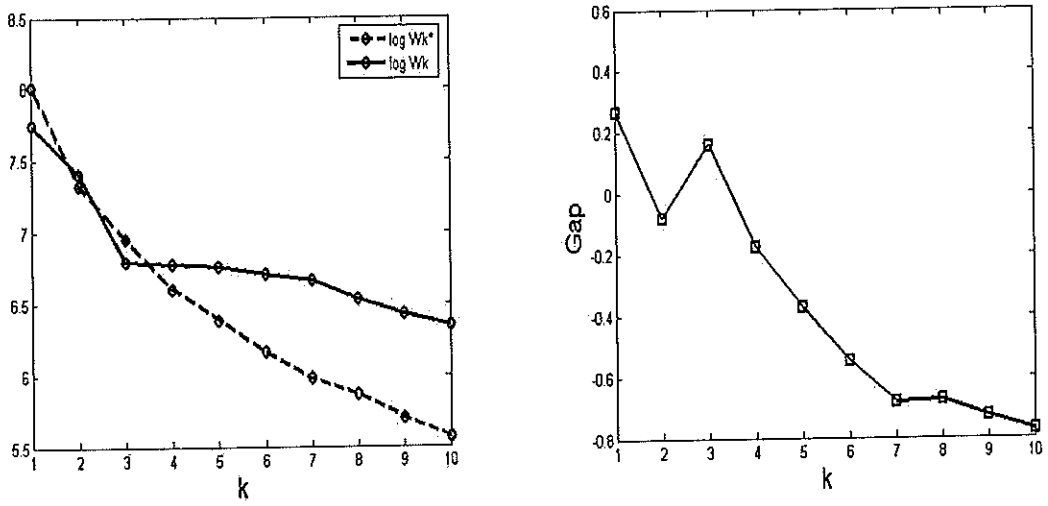


FIG. 4.26 – Output du programme pour le jeu de données triangulaire, sur base d'un partitionnement de cette réalisation par la méthode du lien minimum

Lors de l'exécution de la méthode sur la partition rendue par le lien minimum (figure 4.26), nous assistons à un **effet barrière**. En effet, la valeur de $Gap(2)$ est bien plus petite que celle de $Gap(1)$, d'où la décision portée en $k^\star = 2$. Or, la quantité $Gap(3)$, est largement supérieures à $Gap(2)$. Nous dirons donc que la méthode rencontre une barrière en $k = 2$. Ce phénomène est dû au fait que la classification du lien minimum *ne réduit pas assez* l'inertie intra-classe en partitionnant le jeu de données en *deux classes*. Cette *dépendance de la méthode au seul input qu'est l'inertie intra-classe* est son principal défaut.

4.5.6 Un jeu de données aux classes non convexes

Le jeu de données suivant est formé de 2 classes intuitives non convexes (figure 4.27), croissants imbriqués l'un dans l'autre.

Le tableau d'outputs est le suivant :

	Lien minimum	Lien moyen	Lien maximum	Centroïdes	Ward
	OK				
C-H	2	4	2	2	4
D-H	2	2	2	2	2
CCC	1	1	1	1	1
Gap S.	1	1	1	1	1

Ici, pour toute série hiérarchique de partition, la méthode, tout comme celle du CCC, rend un résultat similaire : $k^\star = 1$. Comme indiqué sur la figure 4.28, toutes les valeurs de Gap sont négatives et $Gap(2) \leq Gap(1)$. Seule la méthode du lien minimum partitionne ce jeu de données en deux classes intuitives.

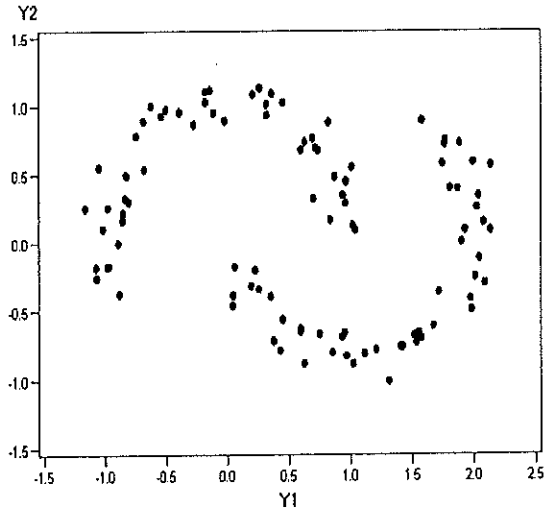


FIG. 4.27 – Un jeu de données formé de classes non convexes

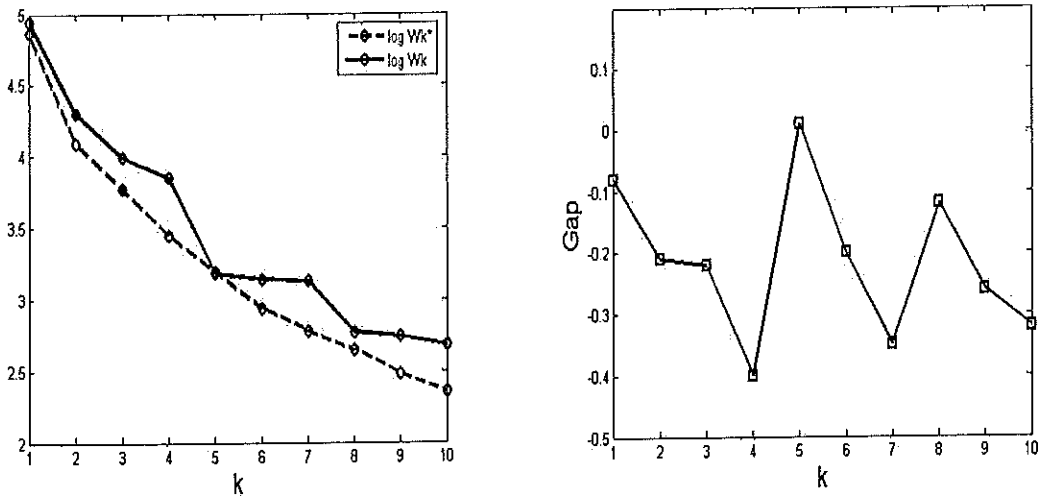


FIG. 4.28 – Output du programme pour le jeu de données aux deux classes non convexes, sur base d'un partitionnement par la méthode du lien minimum

4.6 Conclusion

Le méthode de détermination de classes par la *Gap Statistic* est relativement simple à mettre en oeuvre. Elle ne prend qu'en input les K premières valeurs du R^2 sorties par un algorithme de classification $\mathcal{A}$. Elle prend aussi en compte la bounding box du jeu de données à examiner, afin de générer en ce sous-ensemble de $\mathbb{R}^p$ un processus de Poisson homogène, de taille n.

La méthode a précédemment été testée sur un panel de jeux de données diversifié, et comparée à trois des cinq autres méthodes moins récentes présentées au deuxième chapitre. Dressons donc un tableau récapitulatif.

	C & H	D & H	C	Γ	B	CCC	Hyp	Gap S.
Classes hyperellipsoïdales	X	X	X	X	X	X		+
Données sans structure en classes	X	*qu.*						
Classes de tailles inégales éloignées					X	X		
Classes de tailles inégales proches		X			X	X		
Classes non séparables par un hyperplan		X	X	X	X	X	X	X
Classes non convexes	X	X	X	X	X	X	X	X

Dans ce tableau, les signes + signifient que la méthode fournit des outputs satisfaisants **à partir du moment où l'algorithme de classification sous-jacent partitionne le jeu de données de manière suffisamment intuitive.** Quant aux autres symboles (X, *qu.*, case blanche), ils ont la même signification que celle énoncée à la fin du deuxième chapitre.

En ce qui concerne les résultats sur différentes structures de données, il sera important de souligner que la méthode rivalise aisément avec toutes les méthodes de Milligan et Cooper, mais qu'elle est également capable, au même titre que l'indice CCC, et de par sa construction, de vérifier l'absence de structure en classes. Cependant, cette hypothèse nulle d'absence de structure en classes est difficilement rejetable sur des jeux de données aux classes convexes, ou non séparables par un hyperplan.

L'exécution de la méthode est relativement rapide. La tâche la plus lourde du programme est sans nul doute la génération de B processus de Poisson homogènes de n objets à p composantes. Le reste se concentre essentiellement sur les comparaisons des valeurs $Gap(k)$, au nombre de K dans le pire des cas.

Chapitre 5

Extension au cas symbolique

Dans ce chapitre, nous tâcherons d'étendre la méthode de détermination du nombre de classes vue précédemment dans le cas de variables descriptives classiques, au cas de variables descriptives symboliques, et plus particulièrement, aux **variables intervalles.**

5.1 Définition du cadre symbolique

Dans le cadre classique, une variable Y_j caractérisait un objet $x_i \in E$. Elle donnait une description réelle de x_i :

$$Y_j(x_i) = x_{ij} \in \mathbb{R}.$$

Dans le cadre symbolique, cette description sera plus complexe, plus riche. Parmi les variables symboliques, nous nous intéresserons tout particulièrement aux **variables dites intervalles** [6].

Définition 5.1.1. *Une variable descriptive intervalle (ou descripteur intervalle) sur E est telle que*

$$Y_j : E \longrightarrow \mathcal{Y}, \qquad x_i \rightsquigarrow Y_j(x_i) = [\alpha_{ij}, \beta_{ij}] \subset I\!R,$$

où $\mathcal{Y}$ est l'ensemble de tous les intervalles réels bornés, et donc $-\infty < \alpha_{ij} \leq \beta_{ij} < +\infty$.

Ainsi, au lieu d'associer à un objet de l'ensemble E p descriptions réelles, nous lui attribuerons p intervalles qui caractériseront ce même objet par p bornes minimales et maximales.

Il est aussi possible de créer une distance spécifique à ces objets décrits par p variables intervalles, sur base d'un indice de dissimilarité δ_j entre deux intervalles :

$$D(x_k, x_l) := \left(\sum_{j=1}^{p} \delta_j^2(x_{kj}, x_{lj}) \right)^{\frac{1}{2}}.$$

Voyons les trois formes que peuvent prendre ces indices, avec $x_{kj} = [\alpha_{kj}, \beta_{kj}]$ et $x_{lj} = [\alpha_{lj}, \beta_{lj}]$:

- l'indice δ_1, dit L_1 : $\delta_1(x_{kj}, x_{lj}) = |\alpha_{kj} - \alpha_{lj}| + |\beta_{kj} - \beta_{lj}|$,
- l'indice δ_2, dit L_2 : $\delta_2(x_{kj}, x_{lj}) = (\alpha_{kj} - \alpha_{lj})^2 + (\beta_{kj} - \beta_{lj})^2$,
- l'indice δ_H, dit **de Hausdorff** : $\delta_H(x_{kj}, x_{lj}) = \max\{|\alpha_{kj}, \alpha_{lj}|, |\beta_{kj}, \beta_{lj}|\}$.

5.2 Cadre de classification symbolique

Les partitionnements des jeux de données symboliques "intervalles" réalisés dans ce mémoire seront principalement effectués au moyen du logicial SODAS 2 [28]. L'algorithme de classification choisi sera le module SCLUST [26].

5.2.1 Symbolic dynamic CLUSTering method

Cette méthode est l'extension de la méthode des nuées dynamiques, expliquée au premier chapitre. Elle est itérative et a pour but de déterminer une série de partitions qui améliore à chaque itération la valeur d'un critère mathématique $\mathcal{W}$.

Chacune de ces itérations comprend deux étapes distinctes :

1. une étape de représentation : association à chaque classe d'un **prototype**,
2. une étape d'affectation : affectation des éléments à classer au prototype le plus *proche*.

Le but de la méthode est donc le suivant :

Rechercher :

- $P^\star \in \mathcal{P}_k$, une partition en k classes,
- $L^\star \in \mathcal{L}_k$, un vecteur de k prototypes représentant les classes de P,
  de manière à **minimiser un critère d'ajustement** entre P et L :

$$W(P^\star, L^\star) = \min\{W(P, L) \mid P \in \mathcal{P}_k, L \in \mathcal{L}_k\}.$$

Dans le cas classique, nous avions privilégié la particularisation aux **centres mobiles**, en considérant le prototype d'une classe C_l de la manière suivante :

$$L^{(l)} = \left(\frac{1}{n_l} \sum_{x_i \in C_l} x_{i1}, \ldots, \frac{1}{n_l} \sum_{x_i \in C_l} x_{ip} \right).$$

Le prototype était donc dans ce cas le **centroïde** de la classe.

Dans le cas des intervalles, le prototype sera du même type :

$$L^{(l)} = \left(\left[\frac{1}{n_l} \sum_{x_i \in C_l} \alpha_{i1}, \frac{1}{n_l} \sum_{x_i \in C_l} \beta_{i1} \right], \ldots, \left[\frac{1}{n_l} \sum_{x_i \in C_l} \alpha_{ip}, \frac{1}{n_l} \sum_{x_i \in C_l} \beta_{ip} \right] \right).$$

Ce vecteur que constitue le prototype est appelé **hyperrectangle de gravité** de la classe C_l.

Voyons la manière dont l'algorithme en question est implémenté.

Algorithme des hypperrectangles de gravité mobiles

1. Initialisation
   - Tirer k objets (des **centres-intervalles**) au hasard parmi les n objets de E, construire une partition initiale $P^{(0)} = \{C_1^0, \ldots, C_k^0\}$ en attribuant aux n objets l'indice du centre-intervalle le plus proche, au sens de la distance D, construite sur une dissimilarité δ.
   - Calculer : $n_l := |C_l^0|$ et $L^{(0,l)}$, hyperrectangle de gravité de la classe initiale C_l^0, $\forall\, l \in \{1, \ldots, k\}$.

2. Affectation
   $test := 0$.
   Pour $i = 1$ jusque $i = n$ faire :
   - noter s la classe de x_i,
   - déterminer l tel que
   $$l = \text{argmin}_j \, D^2(x_i, L^{(j)}), \quad \text{où } j \in \{1, \ldots, k\},$$
   - si $l \neq s$ alors
   $$test := 1,$$
   $$C_l := C_l \cup \{x_i\},\ C_s := C_s \setminus \{x_i\},$$
   $$n_l := n_l + 1,\ n_s := n_s - 1.$$

3. Représentation
   Pour $l = 1$ jusque $l = k$ faire :
   calculer l'hyperrectangle de gravité de la nouvelle classe $L^{(l)}$.

4. Si $test \neq 0$ alors retourner à l'étape 2 d'affectation.

5.3 Détermination symbolique du nombre de classes

Grâce à la distance généralisée D, les méthodes de détermination du nombre de classes évoquées au troisième chapitre restent tout à fait utilisables.

Le module NBCLUST sous SODAS 2 a pour but l'application, aux partitions rendues par SCLUST, de trois méthodes de détermination du nombre de classes : Calinski & Harabasz, l'indice C, et l'indice Γ.

Le tableau NBCLUST sera utilisé pour chaque partition sortie au prochain chapitre.

5.4 Représentation Milieu/Longueur

La représentation **Milieu/Longueur** (ML) est appliquée à un jeu de données intervalles E, composé de n objets décrits par p intervalles, particularisé sur une variable Y_d.

Elle peut être vue comme une fonction ML_d définie de la manière suivante :

$$ML_d \ : \ E \longrightarrow \mathbb{R}^2, \quad x_i \rightsquigarrow ML_d(x_i) = \left(\frac{\alpha_{id} + \beta_{id}}{2}, \beta_{id} - \alpha_{id} \right), \qquad [5]$$

où d est l'indice de la variable sur laquelle le jeu de données est particularisé.

Notons que, selon certaines références, c'est la **demi-longueur** qui est privilégiée.

Le but d'une telle représentation, associée à **la plus discriminante des variables intervalles**, est de privilégier une vision ponctuelle du jeu de données, à la base constitué d'individus intervalles.

Exemple d'une représentation ML
Nous allons considérer un jeu de données composé de 20 objets, caractérisés par une seule variable intervalle. Chacun des centres des 20 objets a été simulé, et sa valeur est comprise entre 0 et 10. Deux classes emboîtées sont en réalité distinctes :

1. les 10 premiers intervalles ont été dotés d'une longueur comprise entre 1 et 2,

2. les 10 autres intervalles ont été dotés d'une longueur comprise entre 11 et 12.

Les résultats des modules SCLUST et NBLCUST sous SODAS 2 sont de la forme suivante :

	SCLUST
	OK
C-H	4
C	4
Γ	4

La méthode SCLUST partitionne idéalement le jeu de données, pour $k = 2$. Cependant, comme montré dans le tableau précédent, les méthodes de Calinski & Harabasz, des indices C et Γ fournissent des outputs non cohérents.

L'utilisation d'une représentation ML nous conduit au nuage de points représenté à la figure 5.1

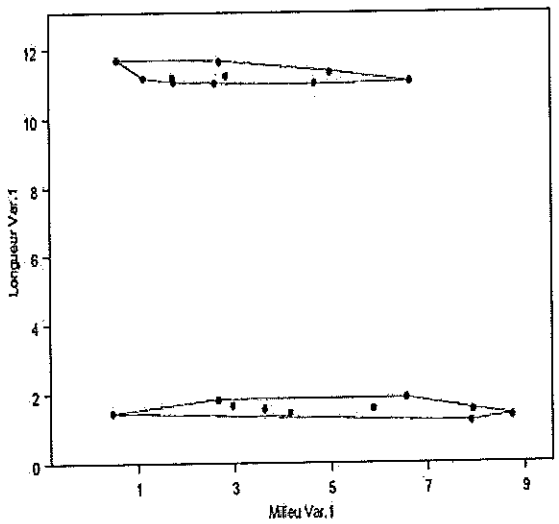


FIG. 5.1 – Représentation ML du jeu de données emboîté aux deux classes délimitées par leur enveloppe convexe

Les deux classes réelles sont ainsi clairement distinctes et leurs prototypes sont représentés à la figure 5.2

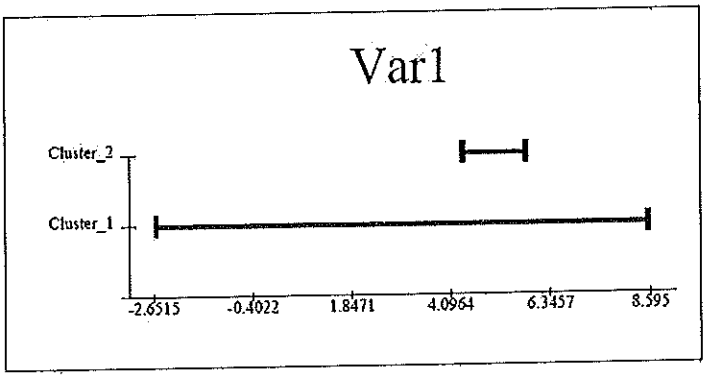


FIG. 5.2 – Prototypes des deux classes emboîtées

Ainsi donc, il sera parfois plus judicieux et plus intuitif de réaliser une classification sur la représentation ML d'un jeu de données intervalles plutôt que sur le jeu de données original.

Déceler la variable la plus discriminante Y_d

Voyons maintenant par quel moyen déceler la variable la plus discriminante Y_d, afin d'obtenir une représentation ML des plus pertinentes.

Supposons le jeu de données partitionné en k classes, selon une dissimilarité δ choisie. Définissons les quantités suivantes :

$$B_j(k) := \sum_{l=1}^{k} n_l \ \delta^2(L_j^{(l)}, L_j),$$

et

$$T_j(k) := \sum_{l=1}^{k} \sum_{x_i \in C_l} \delta^2(x_{ij}, L_j),$$

où L_j est la j^{ime} arête de l'hyperrectangle de gravité du jeu de données entier, et $L_j^{(l)}$ est la j^{ime} arête de l'hyperrectangle de gravité de la classe C_l.

Définissons également la quantité suivante :

$$cor_j(k) := \frac{B_j(k)}{T_j(k)} = \sum_{l=1}^{k} \frac{n_l \ \delta^2(L_j^{(l)}, L_j)}{\sum_{x_i \in C_l} \delta^2(x_{ij}, L_j)}.$$

La variable intervalle Y_d sera la variable la plus discriminante pour le partitionnement de E en k classes si

$$d \ = \ \text{argmax}_{j=1,\ldots,p} \ cor_j(k).$$

5.5 Implémentation de la méthode de la Gap Statistic

Dans le cadre de ce mémoire, deux méthodes ont été implémentées dans le but de déterminer le nombre de classes au sein d'un jeu de données décrit par des intervalles : une généralisation de la méthode étudiée au troisième chapitre, au moyen d'une extension de distance, ainsi qu'une particularisation milieu/longueur.

5.5.1 Extension de distance

La méthode basée sur l'extension de distance constitue une réelle généralisation. En effet, la distance euclidienne d dans le cas classique est remplacée par la distance D, construite sur base d'une dissimilarité δ (L_1, L_2, Hausdorff), dans le cas symbolique intervalle.

Reste la question des B processus de Poisson à générer, dont les logarithmes moyens des inerties intra-classes serviront au calcul de la statistique *Gap*. Afin d'obtenir une distribution la plus uniforme possible d'hyperrectangles d'hypervolumes plus ou moins identiques aux hypervolumes des individus, voyons comment nous avons procédé.

Dans un premier temps, il faudra calculer la bounding box du jeu de données intervalle. La j^{me} arête de cette bounding box, notons-la $A_j := [BB_{j,min}, BB_{j,max}]$, vérifie les égalités suivantes :

$$BB_{j,min} = \min_{x_i \in E} \{\alpha_{ij}\}$$

et

$$BB_{j,max} = \max_{x_i \in E} \{\beta_{ij}\} .$$

Ayant obtenu la bounding box, il faut, sur chaque arête A_j, générer n centres (d'intervalles) c_{ij}, $i \in \{1, \ldots, n\}$, compris dans l'intervalle

$$\left[BB_{j,min} + \frac{\max_{x_i \in E} \{\beta_{ij} - \alpha_{ij}\}}{2}, BB_{j,max} - \frac{\max_{x_i \in E} \{\beta_{ij} - \alpha_{ij}\}}{2} \right] .$$

Il faut également, sur chaque arête A_j, générer n longueurs (d'intervalles) l_{ij}, $i \in \{1, \ldots, n\}$, comprises dans l'intervalle

$$\left[\min_{x_i \in E} \{\beta_{ij} - \alpha_{ij}\}, \max_{x_i \in E} \{\beta_{ij} - \alpha_{ij}\} \right] .$$

Nous poserons enfin :

$$\forall j \in \{1, \ldots, p\}, \ \forall i \in \{1, \ldots, n\}, \quad \alpha^\star_{ij} := c_{ij} - \frac{l_{ij}}{2} \ \text{ et } \ \beta^\star_{ij} := c_{ij} + \frac{l_{ij}}{2} .$$

Nous obtenons ainsi un nouveau jeu de données $E^\star$, constitué d'individus dont les centres et longueurs des intervalles descriptifs ont été générés aléatoirement de telle sorte que les bornes minimales et maximales de ces intervalles **restent dans la bounding box.**

Aussi, remarquons que les longueurs des n intervalles de la variable j sont générées dans l'intervalle dont :

- la borne minimale est la plus petite des longueurs des intervalles $x_{ij} = [\alpha_{ij}, \beta_{ij}]$, $i \in \{1, \ldots, n\}$,
- la borne maximale est la plus grande des longueurs des intervalles $x_{ij} = [\alpha_{ij}, \beta_{ij}]$, $i \in \{1, \ldots, n\}$.

De telles bornes donnent aux hyperrectangles générés $x^\star_i$ des hypervolumes semblables à ceux des individus originaux x_i.

Selon ce modèle, la détermination du nombre de classes, par la *Gap Statistic*, peut être effectuée, au moyen de la distance généralisée D, de manière tout à fait similaire à celle décrite dans le troisième chapitre. Un exemple d'une telle génération selon un modèle nul intervalle est représenté à la figure 5.3.

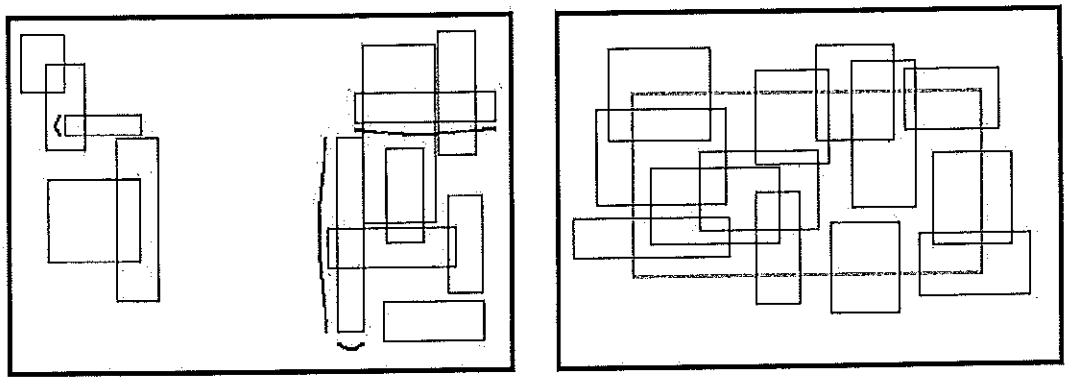


FIG. 5.3 – Exemple de génération d'un modèle nul intervalle adéquat aux centres simulés dans la bounding box gris clair (à droite) sur base d'un jeu de données intervalle original (à gauche), $n = 13$. Les courbures représentent les ranges minimaux et maximaux du jeu de données.

5.5.2 Particularisation milieu/longueur

Ce troisième canevas applique au jeu de données intervalle original une représentation milieu/longueur, selon la variable la plus discriminante ou non. L'algorithme implémenté applique ensuite à ce jeu de données ponctuel bidimensionnel V partitionnements en $k = 1, \ldots, K$ classes par la méthode des centres mobiles, et teste, sur chaque classification réalisée, la règle d'arrêt de la *Gap Statistic*. V décisions quant au nombre de classes sont ainsi prises, et indiquées en output. Les quatre étapes de l'opération sont représentées sur les figures de la page suivante.

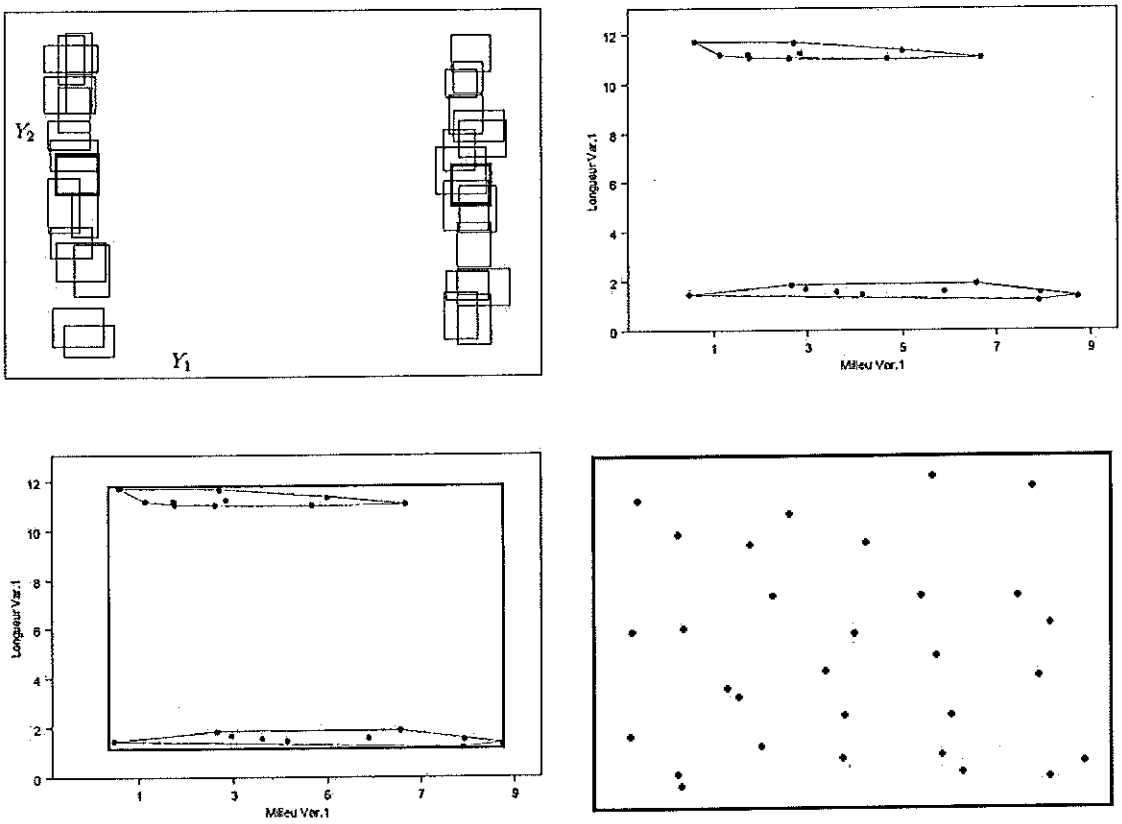


FIG. 5.4 – 1) Un jeu de données intervalle, 2) Représentation ML d'un jeu de données selon la variable la plus discriminante, 3) Calcul de la bounding box de la représentation ML, 4) Génération d'un P. P. H. dans cette bounding box

Chapitre 6

Application aux jeux de données intervalles

Avant d'appliquer les deux méthodes vues au chapitre précédent et de commenter les résultats, le rédacteur conseille vivement la lecture des quatre sections consacrées à l'implémentation de la méthode de la *Gap Statistic* généralisée aux variables intervalles. Ces sections (**A.3**, **A.4**, **A.5**, **A.6**) sont situées en **Annexe A**.

Toutes les applications suivantes sont réalisées au moyen de la distance D basée sur la dissimilarité de Hausdorff δ_H.

Dans ce chapitre d'applications au cas symbolique, nous ne privilégierons cette fois qu'un algorithme de partitionnement décrit au chapitre précédent, SCLUST. Ce choix est porté sur un désir de ne pas alourdir le texte.

6.1 Application à deux jeux de données théoriques

Dans un premier temps, il conviendra de tester les deux méthodes sur des jeux de données dont les structures sont sans équivoque.

6.1.1 Un jeu de données aux classes hypersphériques

Considérons dans un premier temps le jeu de données suivant (figure 6.1), avec $p = 2$, $n = 30$.

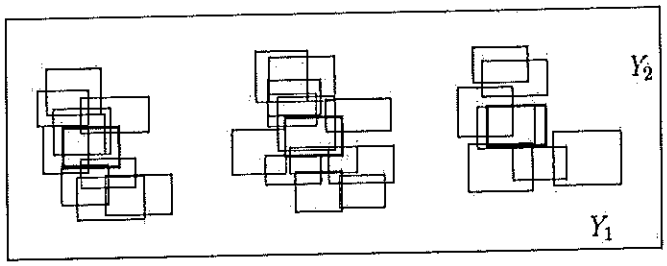


FIG. 6.1 – Jeu de données intervalle aux trois classes hypersphériques distinctes

Le jeu de données est clairement composé de trois classes distinctes. C'est d'ailleurs ces trois classes qui sont retrouvées par la méthode SCLUST, pour $k = 3$.

L'activation du module NBCLUST nous fournit le tableau suivant :

k	C-H	C	Γ
10	110.80336	0.00367	0.92082
9	128.79736	0.00273	0.95515
8	122.08872	0.00412	0.97800
7	137.19192	0.00298	0.97499
6	136.43209	0.00503	0.97035
5	141.36658	0.00612	0.95682
4	150.83094	0.00550	0.97614
3	**174.01620**	**0.00006**	**0.99990**
2	65.29127	0.05465	0.82081
1	-	-	-

Les éléments du tableau marqués en gras sont les quantités pour lesquelles le nombre de classes correspondant est privilégié, conformément à la description des méthodes de détermination du nombre de classes réalisée au deuxième chapitre.

Ainsi, selon NBCLUST, $k^* = 3$, ce qui est intuitivement compréhensible.

Résultat de la méthode obtenue par extension de distance

L'exécution du premier programme implémenté dans un cadre symbolique intervalle nous fournit la variation de la statistique *Gap* suivante (figure 6.2).

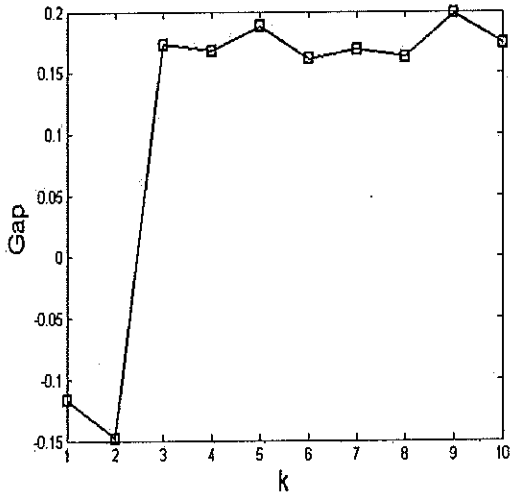


FIG. 6.2 – Variation de la statistique *Gap* en fonction du nombre de classes k, sur base d'un partitionnement réalisé par SCLUST

Bien qu'une croissance très nette de la statistique *Gap* soit marquée de $k = 2$ à $k = 3$, nous assistons à un effet barrière en $k = 2$ ($Gap(2) < Gap(1)$). Ce phénomène est justifié par la construction de la règle d'arrêt de la méthode. En effet, pour $k = 2$, SCLUST divise les données en une classe intuitive pour laquelle les bornes des intervalles Y_1 sont les plus petites, et une classe formée des deux autres sous-groupes intuitifs.

En raison du vide entre ces deux derniers sous-groupes, il est clair que l'inertie intra-classe d'un jeu de données uniforme partitionné en deux classes devrait être plus petite que l'inertie intra-classe du jeu de données original partitionné en les deux classes précédemment données.

Cette méthode conseille donc de privilégier l'assertion $k^\star = 1$. Il s'agit bien sûr d'un mauvais résultat.

Résultat de la méthode obtenue par modélisation ML

Considérons maintenant les deux modélisations milieu/longueur réalisables sur ce jeu de données (figure 6.3).

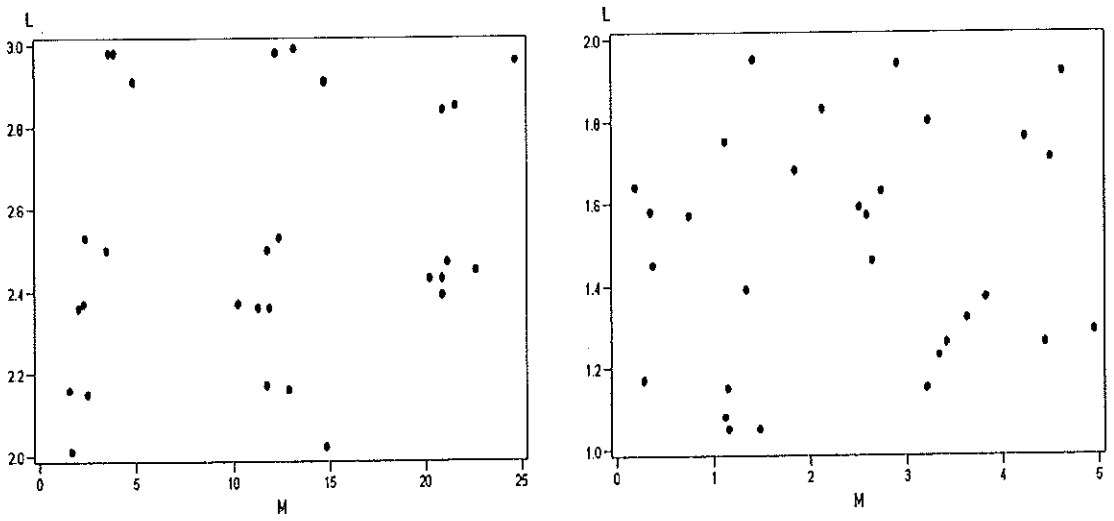


FIG. 6.3 – Représentations ML du jeu de données selon Y_1 (à gauche), selon Y_2 (à droite)

La structure en classes est clairement distincte pour la représentation ML selon la variable Y_1. Y_1 sera donc la variable la plus discriminante. Cependant, la méthode accorde l'assertion $k^\star = 1$ **à 100 % des partitions pour les 2 représentations ML.**

Ce résultat sur cette représentation ML est le strict analogue du paragraphe précédent. Ce type de défaillance, indépendante de l'algorithme de classification sous-jacent, devrait d'ailleurs être incorporée au tableau comparatif en fin du quatrième chapitre. Nous évoquerons ultérieurement ce problème, en émettant avis et idées à propos des artifices utilisés par la méthode.

6.1.2 Un jeu de données aux classes hyperellipsoïdales

Intéressons-nous maintenant au jeu de données suivant, avec $n = 30$, $p = 2$ (figure 6.4).

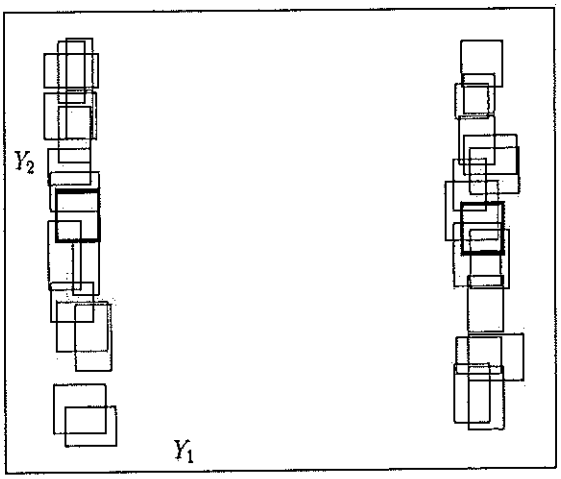


FIG. 6.4 – Jeu de données intervalle aux deux classes allongées distinctes

Le jeu de données est clairement composé de deux classes allongées. Ces deux classes constituent d'ailleurs précisément les deux classes retrouvées par SCLUST, pour $k = 2$.

Voyons d'ailleurs le tableau NBCLUST associé :

k	C-H	C	Γ
10	272.56136	0.00209	0.96597
9	256.58500	0.00268	0.96190
8	194.69276	0.00519	0.98060
7	221.47521	0.00427	0.97978
6	173.24071	0.00922	0.96854
5	187.41755	0.00736	0.96854
4	185.17669	0.00734	0.97689
3	113.97042	0.02663	0.93017
2	**123.07406**	**0.00000**	**1.00000**
1	-	-	-

Selon le module NBCLUST, le nombre de classes optimal est clairement de 2.

Résultat de la méthode obtenue par extension de distance

La méthode basée sur l'extension de distance nous donne les valeurs suivantes pour la statistique *Gap*, en fonction du nombre de classes, représentées sur la figure 6.5.

Bien que la valeur $Gap(3)$ soit plus élevée que la valeur $Gap(2)$, elle ne l'est pas suffisamment pour que la méthode considère des nombres de classes supérieurs à 2. En effet :

$$0.275 = Gap(2) \quad > \quad Gap(3) - \omega \times sd(3) \simeq 0.367 - 1.02 \times 0.102 = 0.26296.$$

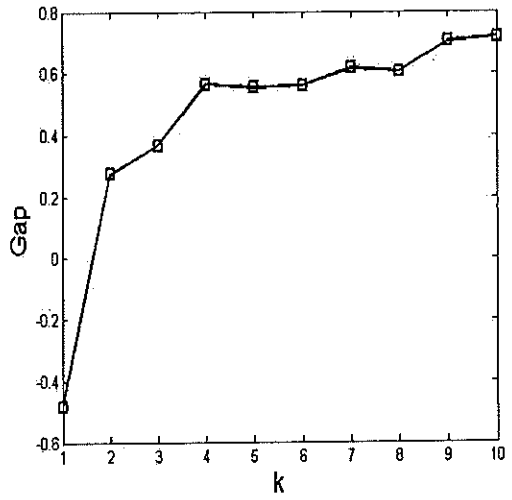


FIG. 6.5 – Variation de la statistique *Gap* en fonction du nombre de classes k, sur base d'un partitionnement réalisé par SCLUST

Cette exécution du programme basé sur l'extension de distance privilégie ainsi l'assertion $k^\star = 2$.

Notons cependant, que pour certaines exécutions du programme, le nombre de classes $k = 4$ est privilégié. Tout dépend de la manière avec laquelle les B jeux de données uniformes sont générés. Précisément, certaines distributions uniformes seront partitionnées de telle sorte que la quantité $Gap(3)$ soit plus élevée que la quantité $Gap(2)$ additionnée à l'écart-type $sd(3)$. D'où $k^\star = 4$. Notons également que cette assertion est, après $k^\star = 2$, la plus privilégiée par le tableau NBCLUST.

Pour trancher sur la question, nous pourrions augmenter considérablement la valeur de l'entier B, afin de se rapprocher d'un modèle uniforme moyen idéal, au sens de l'algorithme défini au troisième chapitre.

Résultat de la méthode obtenue par modélisation ML

La méthode a été exécutée sur les deux représentations ML, issues des variables Y_1 et Y_2 donc (figure 6.6).

La variable Y_1 étant la plus discriminante, la représentation ML selon Y_1 sera celle dont la structure en classes est la plus palpable.

L'autre représentation ML (selon Y_2 donc) n'est d'ailleurs dotée d'aucune structure en classes : 100 % des V partitions en $k = 1, \ldots, K$ classes privilégient l'assertion $k^\star = 1$.

Revenons à la représentation ML selon Y_1. Dans ce cas, 100 % des partitions renvoient $k^\star = 2$. Ce résultat, basé sur une représentation ML plus dicriminante, est intuitivement optimal.

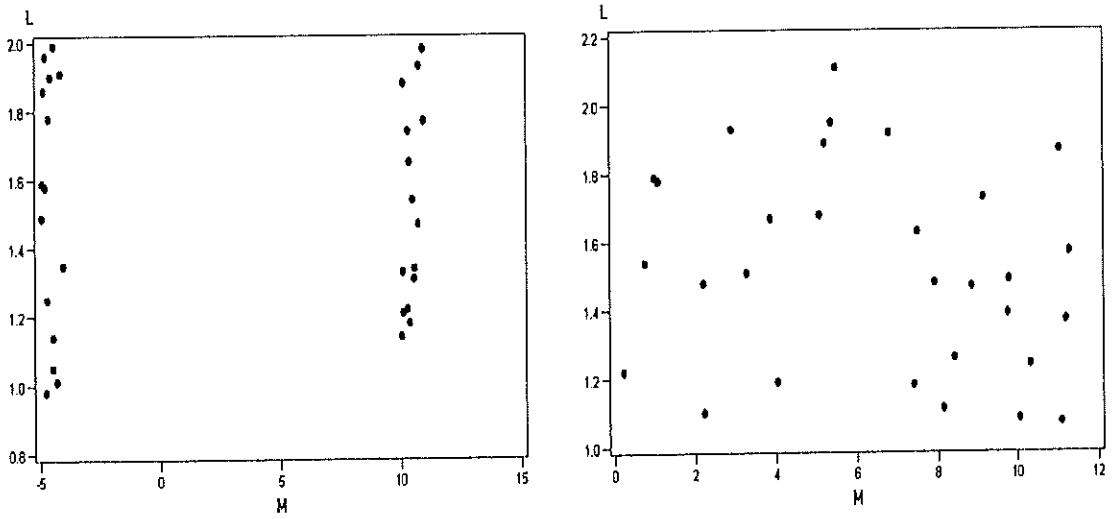


FIG. 6.6 – Représentations ML du jeu de données selon Y_1 à gauche, selon Y_2 à droite

6.2 Application à deux jeux de données sans structure en classes apparente

6.2.1 Un jeu de données sans structure en classes

Soit maintenant un jeu de données intervalle, composé de 50 objets décrits par 3 variables (figure 6.7). Ce jeu de données est généré sans aucune structure en classes.

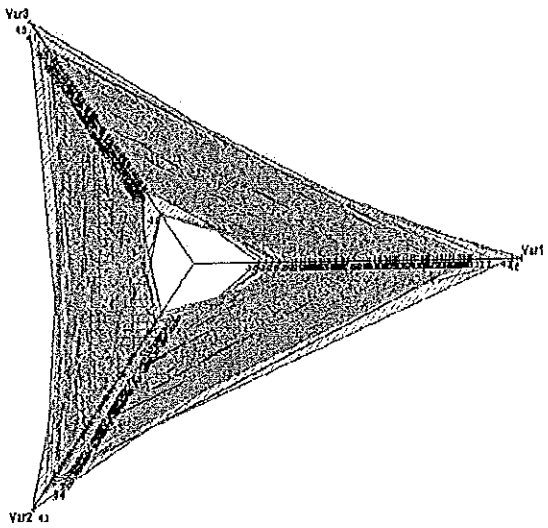


FIG. 6.7 – Zoomstar du jeu de données entier [28]

Voyons le tableau de résultats sorti par le module NBCLUST :

k	C-H	C	Γ
10	8.51267	**0.08582**	**0.72616**
9	9.25804	**0.09590**	**0.69104**
8	7.72520	0.13586	0.61167
7	9.39667	0.11780	0.61691
6	11.79151	**0.09817**	**0.66473**
5	8.81608	0.17752	0.62436
4	9.67400	0.18297	0.59502
3	11.13237	0.22014	0.56881
2	**14.27380**	0.25753	0.53466
1	-	-	-

Les trois méthodes constituant le module NBCLUST ne permettant pas de déceler l'absence de structure en classes, leurs résultats sont non seulement peu intuitifs mais également divergents.

Voyons si, comme au quatrième chapitre consacré aux applications classiques, l'absence de structure en classes est facilement constatable par la méthode.

Résultat de la méthode obtenue par extension de distance

Une exécution du premier programme traitant le cas symbolique intervalle nous fournit l'output suivant (figure 6.8)

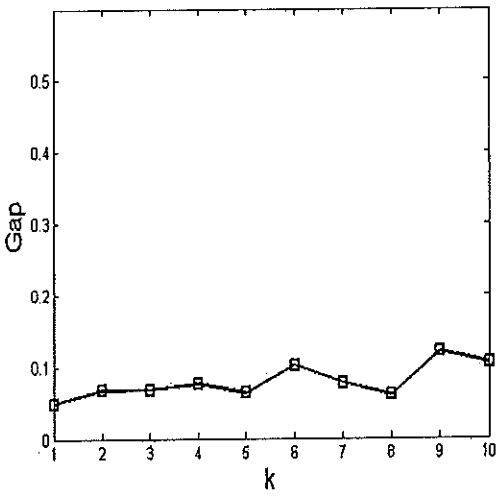


FIG. 6.8 – Variation de la statistique *Gap* en fonction du nombre de classes k, sur base d'un partitionnement réalisé par SCLUST

Les écarts entre les logarithmes des inerties intra-classes du jeu de données uniforme généré et du jeu de données original sont **quasi constants** et **petits**. Logique étant donné

l'absence de structure en classes touchant le jeu de données original, ainsi que la distribution uniforme associée. Ces écarts quasi identiques sont également présents dans le cas classique, comme vu dans le quatrième chapitre (modèles nuls uniformes ou normaux).

L'assertion $k^* = 1$ est donc privilégiée par l'exécution du programme.

Résultat de la méthode obtenue par modélisation ML

Une exécution sur chacune des trois représentations ML induites par les variables Y_1, Y_2 et Y_3 prédit l'absence de structure en classes pour 100 % des partitions.

La structure en classes ($H_1 : k^* > 1$) est donc rejetée par les deux méthodes implémentées, ce à quoi nous pouvions nous attendre en toute logique.

6.2.2 Un jeu de données emboîté

Intéressons-nous à présent à un jeu de données déjà évoqué au chapitre précédent : un jeu de données aux classes intuitives emboîtées. Il s'agit d'un ensemble composé de 20 individus décrits par une seule variable intervalle.

Cette vingtaine d'individus ont cependant été générés de deux manières différentes : chacun des objets intervalles est centré en une valeur comprise entre 0 et 10 mais

1. 10 objets sont de longueur comprise entre 1 et 2,

2. et les 10 autres objets sont de longueur comprise entre 11 et 12.

Il serait donc intuitif de considérer deux classes séparées au sein de ce jeu de données. Voyons ce que suggèrent les artifices mathématiques.

L'algorithme SCLUST donne la classification intuitive, pour $k = 2$ classes. Les prototypes de ces deux classes sont d'ailleurs ici représentés (figure 6.9).

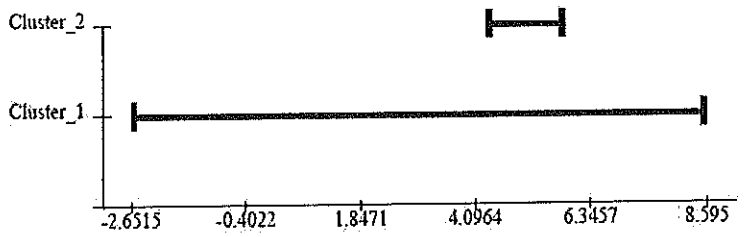


FIG. 6.9 – Prototypes des deux classes retrouvées par SCLUST

Quant au tableau NBCLUST, il est le suivant :

k	C-H	C	Γ
10	124.12694	0.00335	0.96262
9	133.21877	0.00300	0.96571
8	122.02068	0.00292	0.97998
7	127.52085	0.00307	0.99427
6	79.75239	0.01133	0.96044
5	81.84791	0.00646	0.96364
4	**87.50341**	**0.00547**	**0.97354**
3	58.09843	0.02270	0.94068
2	49.07371	0.03008	0.92422
1	-	-	-

Le module NBCLUST privilégie une partition du jeu de données en 4 classes. Ce résultat n'est naturellement pas idéal.

Résultat de la méthode obtenue par extension de distance

L'output obtenu sur base de cette méthode est le suivant (figure 6.10).

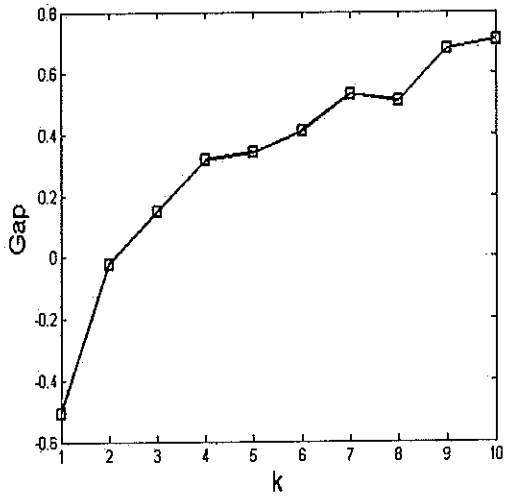


FIG. 6.10 – Variation de la statistique *Gap* en fonction du nombre de classes k, sur base d'un partitionnement réalisé par SCLUST

La méthode conseille l'assertion $k^* = 4$, tout comme le module NBCLUST. Bien que la quantité $Gap(5)$ soit plus grande que la quantité $Gap(4)$, il est clair sur la figure 6.10 que $Gap(5)$ n'est pas assez importante pour considérer des nombres de classes supérieurs à trois :

$$Gap(4) > Gap(5) - \omega \times sd(5).$$

Résultat de la méthode obtenue par modélisation ML

Le représentation ML du jeu de données selon l'unique variable est la suivante (figure 6.11).

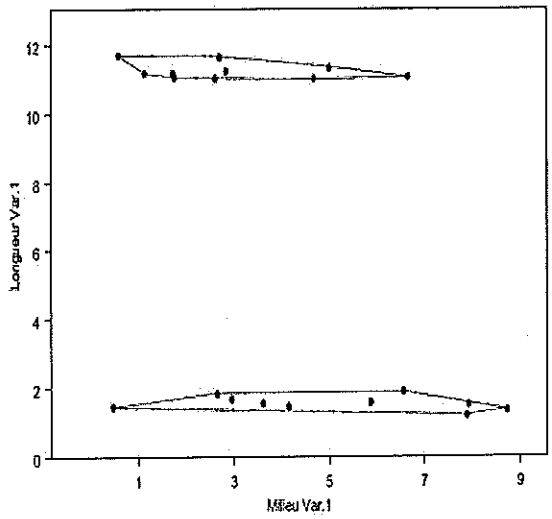


FIG. 6.11 – Représentation ML du jeu de données emboîté

La répartition des nombres de classes évoqués par la méthode est la suivante :

	$k^\star = 2$	$k^\star = 3$	$k^\star = 4$
POURCENTAGE	50.84	23.03	26.13

La méthode basée sur la représentation ML du jeu de données original conseille donc l'assertion intuitive $k^\star = 2$. Notons que la méthode des centres mobiles partitionne facilement cette représentation en les deux classes intuitives initiales.

6.3 Application à deux jeux de données non théoriques

Voyons maintenant les résultats des deux méthodes implémentées sur des jeux de données symboliques intervalles issus de la réalité, aux structures plus complexes donc.

6.3.1 Un jeu de données automobile

Le jeu de données *car.sds* figure parmi les bases du logiciel SODAS 2. Il est composé de 33 marques de voitures décrites par 8 variables intervalles telles que le prix, la vitesse maximale, la cylindrée, etc.

Le tableau NBCLUST est le suivant :

k	C-H	C	Γ
10	56.94598	0.01027	0.81469
9	52.63199	0.01288	0.83460
8	116.36358	0.00763	0.81812
7	62.80247	0.01352	0.84929
6	87.37408	0.00870	0.86421
5	87.05855	0.01369	0.86657
4	57.37063	0.03955	0.80874
3	**73.53980**	**0.01686**	**0.94895**
2	66.97119	0.07725	0.87108
1	-	-	-

NBCLUST conseille clairement l'assertion $k^\star = 3$. La partition associée à ce nombre de classes est constitué de trois gammes de voitures différentes :

1. une **première gamme de voitures de standing moyen**, constituée de modèles tels que Lancia, Nissan, Audi A6, Passat, Focus, ...

2. une **deuxième gamme de voitures de haut standing**, constituée de Porsche, Mercedes, Maserati GT, Audi A8...

3. une **troisième gamme de voitures de très haut standing**, constituée de Ferrari, Aston Martin, Lamborghini, ...

Résultat de la méthode obtenue par extension de distance

Les variations de la statistique *Gap* sont représentées sur le tableau suivant :

k	$Gap(k)$	$sd(k)$
1	-0.529	0.067
2	-0.219	0.079
3	-0.210	0.103
4	-0.155	0.077
5	-0.081	0.079
6	-0.017	0.085
7	-0.117	0.084
8	0.23	0.076
9	-0.154	0.086
10	-0.054	0.088

L'assertion $k^\star = 2$ est privilégiée par la méthode. En effet :

$$-0.219 = Gap(2) > Gap(3) - \omega \times sd(3) = -0.210 - 0.10506 = -0.10494.$$

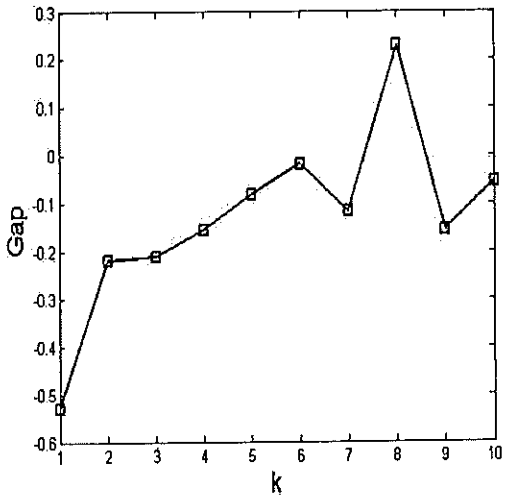


FIG. 6.12 – Variation de la statistique *Gap* en fonction du nombre de classes k, sur base d'un partitionnement réalisé par SCLUST

Ce résultat diffère donc de celui rendu par NBCLUST. La partition en $k = 2$ classes, rendue par NBCLUST, divise les modèles de la même manière, en considérant les deux classes constituées d'une part de la gamme moyenne, et d'autre part des deux classes de haute et très haute gammes.

Résultat de la méthode obtenue par modélisation ML

Nous allons maintenant utiliser le module de discrimination implémenté dans ce troisième programme (voir listing du programme, dernière section de l'annexe).

Nous nous intéresserons aux mesures de discrimination pour les nombres de classes $k = 2$ et $k = 3$.

Ici,

$$\text{argmax}_{j=1,\ldots,p}\ cor_j(2) \;=\; \text{argmax}_{j=1,\ldots,p}\ cor_j(3) \;=\; 1.$$

La variable la plus discriminante est donc la première variable Y_1, le prix du modèle en l'occurrence, ce qui n'est pas étonnant vu les classes retrouvées par SCLUST.

Voyons donc la représentation ML du jeu de données selon Y_1 (figure 6.13)

La méthode, basée sur la représentation ML, affecte l'assertion $k^* = 2$ à chacune des V partitions en $k = 1, \ldots, K$ classes. Cet output peut s'interpréter facilement. En effet, la méthode des centres mobiles sur la représentation ML a tendance séparer **la classe des voitures aux prix bas et peu variables** de la classe des **voitures aux prix moins démocratiques et plus variables** (figure 6.14, page suivante).

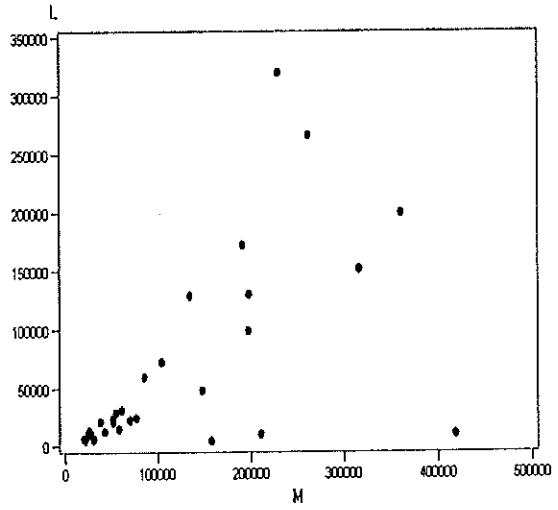


FIG. 6.13 – Représentation ML du jeu de données selon la variable prix Y_1

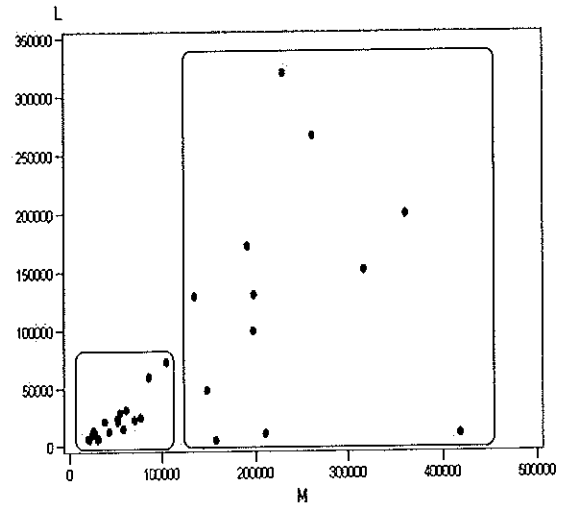


FIG. 6.14 – Représentation ML du jeu de données selon la variable prix Y_1, partitionnée en $k = 2$ classes

6.3.2 Un jeu de données météorologique

Le jeu de données suivant, base du logiciel SODAS 2, est composé de 60 stations chinoises dans lesquelles les températures minimales et maximales en chaque mois ont été mesurées [15]. Ces 60 individus sont donc décrits par 12 variables intervalles.

Le tableau NBCLUST est le suivant :

k	C-H	C	Γ
10	76.55719	0.00736	0.93469
9	86.56895	0.00478	0.95741
8	88.01593	0.00619	0.95314
7	91.71558	0.00905	0.93729
6	90.49053	0.01282	0.91742
5	88.03642	0.02192	0.86976
4	84.52790	0.02980	0.87722
3	77.88730	0.05797	0.80540
2	**95.14143**	0.07597	0.83494
1	-	-	-

Les outputs des trois méthodes constituant le module NBCLUST ne sont pas parfaitement similaires. La méthode C-H semble privilégier sans problème l'assertion $k^\star = 2$ mais les deux autres sont moins franches.

Nous allons donc faire appel au module NBCLUST, cette fois basé sur le module de classification SHICLUST, extension des algorithmes de classification hiérarchique. Leur fonctionnement est strictement similaire à l'algorithme défini au premier chapitre. La seule différence réside dans le fait qu'ils utilisent la distance D distinguant des individus intervalles, et non plus la distance euclidienne d, comme dans le cas de données classiques.

Le module NBCLUST sous SHICLUST nous fournit donc le tableau suivant :

SHICLUST	C-H	D-H	C	Γ	Beale
Lien minimum	$k^\star = 6$	$k^\star = 6$	$k^\star = 6$	$k^\star = 6$	$k^\star = 6$
Lien maximum	$k^\star = 2$	$k^\star = 2$	-	-	$k^\star = 2$
Centroïdes	$k^\star = 2$	$k^\star = 2$	-	-	$k^\star = 2$
Ward	$k^\star = 2$	$k^\star = 2$	-	-	$k^\star = 2$

En conclusion, il est clair que la meilleure partition du jeu de données doit contenir deux ou six classes. L'assertion $k^\star = 2$ revient cependant le plus fréquemment. En réalité, la partition en deux classes est composée d'une classe regroupant les stations balnéaires ou à basse altitude, et d'une classe regroupant les stations situées à l'intérieur des terres, et donc à altitude plus élevée.

Résultat de la méthode obtenue par extension de distance

Voyons si la méthode basée sur l'extension de distance nous fournit des résultats similaires (figure 6.15) :

k	$Gap(k)$	$sd(k)$
1	0.026	0.020
2	0.446	0.021
3	0.597	0.021
4	0.712	0.022
5	0.857	0.023
6	0.932	0.023
7	1.003	0.023
8	1.032	0.023
9	1.054	0.023
10	1.066	0.023

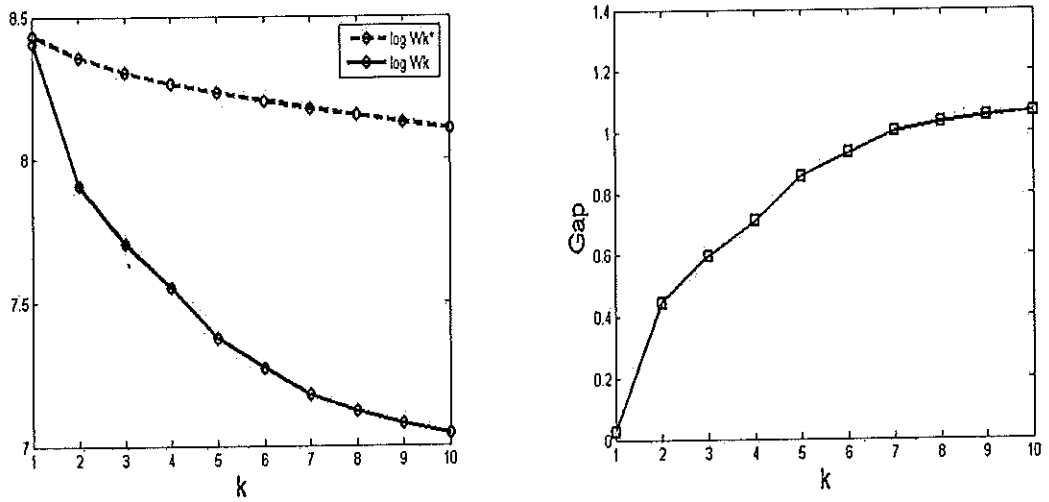


FIG. 6.15 – Variation de la statistique *Gap* en fonction du nombre de classes k, sur base d'un partitionnement réalisé par SCLUST

L'assertion fournie par la méthode est ici $k^* = 8$. Cependant, aucun pic n'est décelable sur la figure. La règle d'arrêt est d'ailleurs évaluée un trop grand nombre de fois. Nous considérerons donc que la méthode ne fournit pas un résultat erroné, mais qu'elle est inutilisable sur le jeu de données en question.

6.4 Conclusion

Globalement, la méthode de la *Gap Statistic*, étendue au cas symbolique, est très loin d'être mauvaise. Nous pouvons, malgré ses défauts intrinsèques (retrouvables dans le cas de données classiques), lui accorder nombre d'avantages.

Le premier réside dans sa capacité à retrouver, contrairement à la quasi totalité des méthodes dont nous disposions auparavant, l'absence de structure en classes. Cette caractéristique a été prouvée dans le cas classique, comme dans le cas symbolique (intervalle). Preuve en a été faite à la sous-section 6.2.1.

En ce qui concerne la sous-section 6.1.1, même si le test d'arrêt de la méthode **privilégie les maxima locaux** de la statistique *Gap*, il suffira simplement de jeter un oeil aux différentes valeurs de celle-ci pour admettre que, pratiquement, il faudrait parfois dépasser le voisinage en question afin de trouver un nombre de classes $k^\star$ plus adapté.

En conclusion, force est de constater que la méthode rivalise aisément avec les autres méthodes, et qu'elle a le très net avantage de pouvoir déceler l'absence de structure en classes.

Quant à la manière de générer des jeux de données symboliques intervalles uniformes, elles n'ont été que tentative de étendre honnêtement le processus de Poisson homogène dans le cas des données classiques. Certes, il serait possible de générer ces jeux de données uniformes symboliques sous-jacents sous d'autres hypothèses, au moyen d'autres artifices non évoqués ici.

Notons également que ce chapitre a été réalisé exclusivement sous la distance D créée à partir de la dissimilarité de Hausdorff (δ_H). Nous pourrions également tester la méthode dans des cadres délimités par les distances L_1 et L_2.

Conclusion et perspectives

La méthode de la Gap Statistic est une méthode de détermination du nombre de classes dont l'avantage majeur est sa capacité à caractériser une absence de structure en classes pour des données quantitatives classiques.

Elle semble être capable d'exécuter cette tâche également dans le cas symbolique intervalle. Du moins dans le cadre construit à la fin de ce document, construit sur base d'arguments non justifiés théoriquement, rappelons-le.

Ses résultats, étudiés le plus complètement aux quatrième et sixième chapitres, sont également globalement acceptables sur des jeux de données intuitifs.

Cependant, il nous semble nécessaire d'émettre opinions et perspectives dans le but d'une amélioration de cette méthode. Notamment quant aux points suivants.

- La **bounding box** du jeu de données original dans laquelle est généré le processus de Poisson : sur base du résultat énoncé par B. Ripley & J.-P. Rasson (1977), ne devrions-nous pas réduire cette bounding box à l'enveloppe convexe dilatée du jeu de données, dont elle serait l'estimateur non biaisé sous l'hypothèse nulle ? Un tel modèle permettrait de mettre en évidence les écarts entre inerties (originales et simulées). Cette approche a d'ailleurs déjà utilisée, dans d'autres cadres de détermination du nombre de classes, par Bertrand & Bel Mufti (2006).

- L'**intensité du processus de Poisson** généré : doit-elle être définie en fonction du nombre d'individus, ou de l'hypervolume de la bounding box du jeu de données original ? Un outsider augmente plus facilement l'hypervolume de la bounding box que l'enveloppe convexe du jeu de données en question.

- La **règle d'arrêt**, et sa **constante de relaxation** ω : sont-elles justifiées ? N'oublions pas que la méthode de la Gap Statistic n'est qu'un formalisme mathématique de la méthode du coude. Certes la méthode réagit de bonne manière sur les jeux de données sans structure en classes. Elle a cependant prouvé, notamment sur les deux premiers jeux de données du sixième chapitre, qu'elle avait parfois tendance à rejeter $H_1 : k^* > 1$, tantôt trop vite, tantôt trop tard. Il serait sans doute intéressant de baser une règle d'arrêt uniquement sur les croissances de la statistique *Gap*.

Annexe A

Programmation de la méthode et de ses extensions

A.1 Implémentation de la méthode dans le cas classique

Cette section a pour but de commenter succinctement la manière dont fonctionne le code (**gap.c** en langage C) implémenté selon l'algorithme du troisième chapitre. Les codes, commentés en détails, seront repris dans cette annexe. Ils sont visualisables sous n'importe quel compilateur C ou C++. Ce chapitre procède par énumération de *commandes*, repérées dans ce même code et correspondant à une exécution particulière. Notons que cette première section concerne le cas classique. Les extensions symboliques du programme sont commentées ultérieurement.

Les codes et exécutables sont disponibles sur un package accompagnant ce mémoire.

A.1.1 Premier bloc : lecture de l'input

Voyons quel type d'input prend en compte le programme **gap.c**. Il s'agit d'un fichier texte de la forme suivante.

```
15
3

14      76      8.9
13      90      9
12.14   22      67
67      88      14
110     45.667  21
76      23.1    2.19
89      98.02   12
22      43      0.8
29.2    100.1   43.567
101     99      21.59
69      61      83
34.8    37      39
1.13    110     45
12      108     22.2
13.33   16.89   90.11

0
0.2934028055
0.6154358634
0.6852862285
```

```
    0.7811946565
25  0.8870412215
    0.9085738709
    0.9262672208
    0.9598825289
    0.9727123032
```

Cet input est constitué de trois parties : la première comprend l'entier $n = 15$ (la taille de E, ligne 1) et l'entier $p = 3$ (la dimension de E, ligne 2), la deuxième comprend les $n \times p = 15 \times 3 = 45$ descriptions x_{ij} (la matrice des données classiques de E, lignes 4-18), tandis qu'à la troisième et dernière partie sont associées les K premières valeurs pour la statistique R^2 (lignes 20-29), rendues par un algorithme de classification $\mathcal{A}$ fixé (sous une application SAS, par exemple).

Le premier bloc du code consiste essentiellement à ouvrir en lecture le fichier **exemple.txt**, à lire les taille, dimension et descriptions de E, et les K premières valeurs du coefficient R^2 rendues par l'algorithme renseigné $\mathcal{A}$ (*commandes 2-5*).

Ce bloc initialise également trois constantes très utiles (*commande 1*) :

1. le nombre maximal de classes à considérer : K, fixé à $K = 10$ au troisième chapitre,

2. le nombre de processus de Poisson à générer : B, fixé à $B = 25$ précédemment,

3. la valeur de la constante de rejet évoquée au troisième chapitre :

$$\omega = \sqrt{1 + \frac{1}{B}} = \sqrt{1.04} = 1.019804.$$

A.1.2 Deuxième bloc : la boucle GAP

Le programme génère avant tout $B = 25$ processus de Poisson dans la bounding box du jeu de données (*commande 8*). Il calcule également les logarithmes des inerties intra-classes extraites des R^2 insérés dans l'input (*commandes 6-7*). Il calcule de la même manière le vecteur reprenant les logarithmes moyens des inerties intra-classes rendus par la méthode des centres mobiles appliquée aux processus de Poisson (*commande 9*). Ces logarithmes sont stockés dans une matrice temporaire, afin de pouvoir ensuite calculer leurs écarts-types, comme le conseille la méthode (*commande 11*). Les estimations de la statistique *Gap*, selon le nombre de classes $k \in \{1, \ldots, K\}$, sont ensuite réalisées (*commande 10*). Le "bon" nombre de classes au sens de la méthode est ensuite décidé (*commande 12*).

A.1.3 Troisième bloc : le calcul des K inerties intra-classes

Ce bloc est largement commenté au sein du programme (*commande 13*). Il s'agit d'une procédure qui prend en input la matrice associée au jeu de données (ainsi donc que ses taille et dimension), et le nombre maximal K de classes à obtenir. Il rend donc un vecteur constitué de K composantes, les logarithmes des inerties intra-classes du jeu de données partitionné en $1, \ldots, K$ classes.

A.1.4 Output

L'output est constitué des valeurs estimées des statistiques *Gap*, ainsi que des écarts-types des logarithmes des inerties intra-classes des B processus de Poisson. La décision de la méthode est également indiquée.

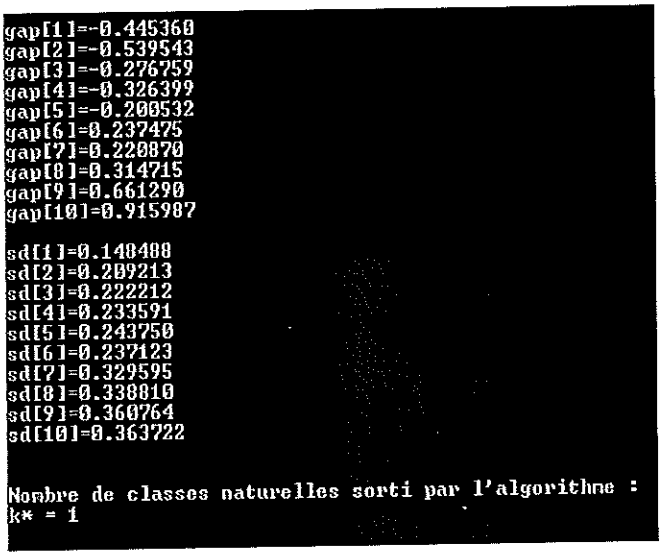


FIG. A.1 – Output de la méthode sur le jeu de données précédent

A.1.5 Header du code principal : *gap.h*

```
/* HEADER ATTACHE AU PROGRAMME gap.c */

double *intraI(double **Y,int KK,int nn,int pp);

double dist(double *arg1,double *arg2,int pp);
```

A.1.6 Listing du code principal : *gap.c*

```
/* DETERMINATION DU NOMBRE DE CLASSES AU MOYEN DE LA GAP STATISTIC
(CAS CLASSIQUE) */

/*** Auteur : S. Cosme, 2007, F. U. N. D. P.
PROMOTEUR DE MEMOIRE : A. Hardy ***/

#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include "gap.h" //header attaché

main ()
{

     /*******************************/
     /* BLOC I : LECTURE DE L'INPUT */
     /*******************************/

     printf("DETERMINATION DU NOMBRE DE CLASSES PAR LA GAP STATISTIC\n");
     printf("(cas classique)\n");
     printf("============================================================\n\n");

     //déclaration de compteurs entiers
     int I,II,III;

     //CMDE 1 : déclaration de constantes utiles
     int B,K;

     B=25;
     K=10;

     //calcul de la constante de rejet
     double W;

     W=B;
     W=sqrt(1+(1/W));

     //CMDE 2 : ouverture du fichier "txt" input en mode lecture
     char *finput;

     finput=calloc(30,sizeof(char));

     printf("FICHIER TXT D'INPUT : ");
     scanf("%s",finput);

     FILE *fichier=NULL;
     fichier=fopen(finput,"r");

     free (finput);

     //CMDE 3 : lecture des taille et dimension du jeu de données original
     int n,p;

     fscanf(fichier,"%d %d",&n,&p);

     //réduction du nombre de classes maximal à un entier inférieur à la taille
     //du jeu de données original si nécessaire
```

```
        if (K>=n)
60      {K=n-1;}

        //CMDE 4 : lecture de la matrice des données classiques du jeu de données
        //original
        double **X;
65
        X=calloc(n,sizeof(double*));
        for (I=0;I<n;I++)
        {X[I]=calloc(p,sizeof(double));}

70      for (I=0;I<n;I++)
        {
            for (II=0;II<p;II++)
            {fscanf(fichier,"%lf",&X[I][II]);}
        }
75
        //CMDE 5 : lecture des Rš sur l'input
        double *rcarre;

        rcarre=calloc(K,sizeof(double));
80
        for (I=0;I<K;I++)
        {fscanf(fichier,"%lf",&rcarre[I]);}

        //fermeture du fichier d'input
85      fclose(fichier);
        free(fichier);

        /***************************/
        /* BLOC II : LA BOUCLE GAP */
90      /***************************/

        //CMDE 6 : stockage de l'inertie totale du jeu de données original
        double inertietotale;

95      inertietotale=0;

        for (I=0;I<n;I++)
        {
            for (II=0;II<n;II++)
100         {inertietotale=inertietotale+dist(X[I],X[II],p)/(2*n);}
        }

        //CMDE 7 : calcul des logarithmes des inerties intra-classes
        //du jeu de données original
105     double *orig;

        orig=calloc(K,sizeof(double));

        for (I=0;I<K;I++)
110     {orig[I]=log(inertietotale*(1-rcarre[I]));} //car W = T * (1-Rš)

        //calcul des bornes de la bounding box du jeu de données original
        double *min,*max;

115     min=calloc(p,sizeof(double));
        max=calloc(p,sizeof(double));

        for (I=0;I<p;I++)
        {
120         min[I]=X[0][I];
            max[I]=X[0][I];
            for (II=0;II<n;II++)
            {
                if (min[I]>X[II][I])
125             {min[I]=X[II][I];}
                if (max[I]<X[II][I])
```

```
                {max[I]=X[II][I];}
            }
        }
130
        //génération d'un entier aléatoire rendue possible
        srand (time(NULL));

        //CMDE 8 : génération dans la bounding box des processus de Poisson
135     double ***poisson;

        poisson=calloc(B,sizeof(double**));
        for (I=0;I<B;I++)
        {
140          poisson[I]=calloc(n,sizeof(double*));
             for (II=0;II<n;II++)
             {poisson[I][II]=calloc(p,sizeof(double));}
        }

145     for (I=0;I<B;I++)
        {
            for (II=0;II<n;II++)
            {
                for (III=0;III<p;III++)
                {
150               poisson[I][II][III]=rand();
                  poisson[I][II][III]=poisson[I][II][III]/RAND_MAX;
                  poisson[I][II][III]=poisson[I][II][III]*(max[III]-min[III]);
                  poisson[I][II][III]=poisson[I][II][III]+min[III];
155             }
            }
        }

        //CMDE 9 : calcul des logarithmes des inerties intra-classes des processus
160     //de Poisson
        double *artif,**tempw;

        artif=calloc(K,sizeof(double));

165     tempw=calloc(B,sizeof(double*));
        for (I=0;I<B;I++)
        {tempw[I]=calloc(K,sizeof(double));}

        for (I=0;I<K;I++)
170     {artif[I]=0;}

        for (I=0;I<B;I++)
        {
            tempw[I]=intraI(poisson[I],K,n,p);
175
            for (II=0;II<K;II++)
            {artif[II]=artif[II]+((tempw[I][II])/B);}
        }

180     //CMDE 10 : calcul du vecteur "GAP"
        double *gap;

        gap=calloc(K,sizeof(double));

185     for (I=0;I<K;I++)
        {gap[I]=artif[I]-orig[I];}

        //CMDE 11 : calcul des écarts-types des logarithmes des
        //inerties intra-classes des processus de Poisson
190     double *sd;

        sd=calloc(K,sizeof(double));

        for (I=0;I<K;I++)
```

```
195     {
            sd[I]=0;
            for (II=0;II<B;II++)
            {sd[I]=sd[I]+((tempw[II][I]-artif[I])*(tempw[II][I]-artif[I])/B);}
            sd[I]=sqrt(sd[I]);
200     }

        //CMDE 12 : BOUCLES GAP
        for (I=0;I<K;I++)
        {
205         printf("artif[%d]=%lf\n",I+1,artif[I]);
            printf("orig[%d]=%lf\n\n",I+1,orig[I]);
        }
        printf("\n");

210     for (I=0;I<K;I++)
        {printf ("gap[%d]=%lf\n",I+1,gap[I]);}
        printf("\n");

        for (I=0;I<K;I++)
215     {printf("sd[%d]=%lf\n",I+1,sd[I]);}

        //---output de l'exécution pure---
        I=0;
        while (gap[I]<gap[I+1]-(sd[I+1]*W))
220     {I++;}
        if (gap[I]>=gap[I+1]-(sd[I+1]*W))
        {
           printf("\n\nNombre de classes naturelles sorti ");
           printf("par l'algorithme :\nk* = %d\n\n",I+1);
225     }

        //désallocation générale de mémoire
        for (I=0;I<n;I++)
        {free (X[I]);}
230     free (X);

        free (rcarre);

        free (orig);
235
        free (min);

        free (max);

240     for (I=0;I<B;I++)
        {
            for (II=0;II<n;II++)
            {free (poisson[I][II]);}
            free (poisson[I]);
245     }
        free (poisson);

        free (artif);

250     free (gap);

        free (sd);

        for (I=0;I<B;I++)
255     {free (tempw[I]);}
        free (tempw);

        //arrêt du programme
        printf("\n\n\n");
260     system ("PAUSE");
        return (0);
}
```

```
/*****************************/
/* BLOC III : SOUS-ROUTINES */
/*****************************/

/***** CMDE 13 *****/
//CALCUL DE L'INERTIE INTRA-CLASSES d'un jeu de données Y partitionné (de
//taille nn et de dimension pp), en 1 <= kk <= KK classes, sur base de la méthode
//des centres mobiles

double *intraI(double **Y,int KK,int nn,int pp)
{
        int haz,haz2;
        int h,i,j,k;
        int test,place;
        int *card,*aff;

        card=calloc(KK,sizeof(int));

        aff=calloc(nn,sizeof(int));

        double distancemin;
        double **centre,**prak;

        centre=calloc(KK,sizeof(double*));
        for (k=0;k<KK;k++)
        {centre[k]=calloc(pp,sizeof(double));}

        prak=calloc(nn,sizeof(double*));
        for (i=0;i<nn;i++)
        {prak[i]=calloc(KK,sizeof(double));}

        double *rez;
        rez=calloc(KK,sizeof(double));

        //cas de l'inertie totale (k=1)
        rez[0]=0;
        for (i=0;i<nn;i++)
        {
            for (j=0;j<nn;j++)
            {rez[0]=rez[0]+dist(Y[i],Y[j],pp)/(2*nn);}
        }
        rez[0]=log(rez[0]);

        //initialisation des KK centres DIFFERENTS
        for (k=0;k<KK;k++)
        {
           if (k==0)
           {haz=(rand() % nn);}
           else
           {
              do
              {haz2=(rand() % nn);}
              while (haz2==haz);
              haz=haz2;
           }
           for (j=0;j<pp;j++)
           {centre[k][j]=Y[haz][j];}
        }

        for (h=1;h<KK;h++)
        {
              //initialisation de la table d'affectation
              for (i=0;i<nn;i++)
              {aff[i]=0;}

              //exécution de la méthode des centres mobiles
              test=1;
```

```
        while (test==1)
        {
              test=0;

              //calcul des distances au carré séparant les objets
              //des centres actuels
              for (i=0;i<nn;i++)
              {
                  for (k=0;k<=h;k++)
                  {prak[i][k]=dist(Y[i],centre[k],pp);}
              }

              //réinitialisation des cardinaux des classes potentielles
              for (k=0;k<=h;k++)
              {card[k]=0;}

              //affectation sur base des distances
              //séparant objets et centres
              for (i=0;i<nn;i++)
              {
                  distancemin=prak[i][0];
                  place=0;
                  for (k=0;k<=h;k++)
                  {
                      if (distancemin>prak[i][k])
                      {
                         distancemin=prak[i][k];
                         place=k;
                      }
                  }
                  if (place!=aff[i])
                  {test=1;}

                  aff[i]=place;
                  card[aff[i]]++;
              }

              //mise à jour des centres
              for (k=0;k<=h;k++)
              {
                 for (j=0;j<pp;j++)
                 {centre[k][j]=0;}
              }
              for (i=0;i<nn;i++)
              {
                 for (j=0;j<pp;j++)
                 {centre[aff[i]][j]=centre[aff[i]][j]
                 +(Y[i][j]/card[aff[i]]);}
              }
         }

        //calcul de l'inertie intra-classe résultante
        rez[h]=0;
        for (i=0;i<nn;i++)
        {rez[h]=rez[h]+dist(Y[i],centre[aff[i]],pp);}
        rez[h]=log(rez[h]);
    }

    return rez;

    //désallocation de mémoire interne à la procédure
    free (card);

    free (aff);

    for (i=0;i<KK;i++)
    {free (centre[i]);}
    free (centre);
```

```
400         for (i=0;i<nn;i++)
            {free (prak[i]);}
            free (prak);

            free (rez);
405 }

    //procédure calculant une distance au carré entre deux vecteurs
    double dist(double *arg1,double *arg2,int pp)
    {
410         double rez;

            int i;

            rez=0;
415
            for (i=0;i<pp;i++)
            {rez=rez+(arg1[i]-arg2[i])*(arg1[i]-arg2[i]);}

            return rez;
420 }
```

A.2 Implémentation de la méthode dans le cas symbolique intervalle

Nous avons implémenté, en langage C, les deux sous-méthodes décrites au cinquième chapitre. Dans les deux cas, l'input est de la forme suivante.

```
30
2

3.37    6.28    -0.35   1.10
0.81    3.17    3.35    5.11
2.19    4.69    0.57    1.72
1.06    3.59    1.92    3.55
2.33    5.31    2.72    3.95
0.63    2.64    1.21    3.04
0.43    2.59    2.96    4.28
2.10    5.08    -0.63   1.01
1.38    3.53    -0.04   1.53
1.05    3.42    1.91    3.37
13.21   16.12   2.63    3.78
10.07   12.43   3.66    5.58
10.46   12.96   0.58    1.66
11.05   13.58   1.93    3.87
10.66   13.64   3.64    5.35
13.79   15.81   -0.30   0.87
10.60   12.77   2.78    4.04
11.63   14.62   0.95    2.00
11.77   13.93   -0.44   1.14
9.02    11.39   1.00    2.68
13.27   16.18   0.64    2.03
10.64   13.00   3.13    4.50
19.68   22.07   4.30    5.59
21.39   23.84   0.63    1.68
20.14   22.99   1.71    3.30
19.64   22.07   3.81    5.07
19.01   21.44   2.32    4.12
23.21   26.17   0.43    2.38
19.48   22.32   0.24    1.99
19.90   22.37   1.80    3.37

0
0.4130
0.6767
0.7313
0.7742
0.7988
0.8160
0.8266
0.8465
0.8552

50.29
82.30
86.17
87.67
86.07
90.01
90.20
90.73
91.49

0
0.44
13.19
30.33
51.41
42.93
```

```
48.00
56.68
58.04
```

Les entiers 30 et 2, aux lignes 1 et 2, constituent les taille et dimension du jeu de données intervalles.
Quant aux quatre colonnes, des lignes 4 à 33, il s'agit des composantes de la matrice des données symboliques. En effet, les deux premières colonnes $[3.37, \ldots, 19.90]$ et $[6.28, \ldots, 22.37]$ sont en réalité les colonnes reprenant les minima et maxima des 30 descriptions réalisées par la variable Y_1 sur tous les individus. Quant aux deux autres colonnes, il s'agit des colonnes reprenant les minima et maxima des 30 descriptions réalisées par la variable Y_2 sur tous les individus.
Les pourcentages (aux lignes 35 à 44) constituent les valeurs des K premiers R^2 fournis par la méthode SCLUST sous SODAS.
Quant aux deux dernières colonnes, elles représentent les quantités $cor_j(k)$, mesures définies au cinquième chapitre, ayant pour but de repérer la variable la plus discriminante dans le cas d'une modélisation milieu/longueur.

A.2.1 Implémentation de la méthode obtenue par extension de distance

Le programme qui suit est très similaire au programme précédent. La différence principale réside dans la distance utilisée entre individus. Cette distance est redéfinie à la ligne 493 du code, selon la dissimilarité choisie à la ligne 59.

Notons également que, pour tout $j \in \{1, \ldots, p\}$, pour tout $i \in \{1, \ldots, n\}$, la j^{me} composante du i^{me} individu est cette fois représentée par un vecteur de $\mathbb{R}^2$ (voir commandes 4 et 8).

Enfin remarquons maintenant que les modèles nuls générés selon les directives du cinquième chapitre sont cette fois partitionnés par la méthode des hyperrectangles de gravité mobiles (voir bloc 3).

Notons que cette méthode construite sur une extension de distance ne prend pas en input les valeurs des mesures de discrimination $cor_j(k)$, mais bel et bien les R^2.

L'output du programme dans le cas de l'input exposé précédemment est semblable à celui fourni par le programme traitant le cas classique. Les K valeurs de la statistique Gap ainsi que les K écarts-types sont affichés. La décision quant au nombre de classes est également inscrite.

Header du code principal : *gap2.h*

```
/* HEADER ATTACHE AU PROGRAMME gap2.c */

double *intraI(double ***Y,int KK,int nn,int pp,int diss);

double dist(double **arg1,double **arg2,int pp,int diss);

double maximum(double arga,double argb);
```

Listing du code principal : *gap2.c*

```
/* DETERMINATION DU NOMBRE DE CLASSES AU MOYEN DE LA GAP STATISTIC
(CAS SYMBOLIQUE-INTERVALLE), extension de distance */

/*** Auteur : S. Cosme, 2007, F. U. N. D. P.
PROMOTEUR DE MEMOIRE : A. Hardy ***/

#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include "gap2.h" //header attaché

main ()
{

     /*******************************/
     /* BLOC I : LECTURE DE L'INPUT */
     /*******************************/

     printf("DETERMINATION DU NOMBRE DE CLASSES PAR LA GAP STATISTIC\n");
     printf("(cas symbolique-intervalle), extension de distance\n");
     printf("=========================================================\n\n");

     //déclaration de compteurs entiers
     int I,II,III;

     //CMDE 1 : déclaration de constantes utiles
     int B,K;

     B=2500;
     K=10;

     //calcul de la constante de rejet
     double W;

     W=B;
     W=sqrt(1+(1/W));

     //CMDE 2 : ouverture du fichier "txt" input en mode lecture
     char *finput;

     finput=calloc(30,sizeof(char));

     printf("FICHIER TXT D'INPUT : ");
     scanf("%s",finput);

     FILE *fichier=NULL;
     fichier=fopen(finput,"r");

     free (finput);

     //CMDE 2b : choix de la dissimilarité entre intervalles
     int dissim;

     printf("\n\nAvant toute chose, entrez la dissimilarité\navec laquelle\n");
     printf("vous voulez travailler\n\n1 pour L1\n2 pour L2\n3 pour Hausdorff");
```

```
printf("\n\n");

scanf("%d",&dissim);
printf("\n\n");

//CMDE 3 : lecture des taille et dimension du jeu de données original
int n,p;

fscanf(fichier,"%d %d",&n,&p);

//réduction du nombre de classes maximal à un entier inférieur à la taille
//du jeu de données original si nécessaire
if (K>=n)
{K=n-1;}

//CMDE 4 : lecture de la matrice des données symboliques du jeu de données
//original
double ***X;

X=calloc(n,sizeof(double**));
for (I=0;I<n;I++)
{
    X[I]=calloc(p,sizeof(double*));
    for (II=0;II<p;II++)
    {X[I][II]=calloc(2,sizeof(double));}
}

for (I=0;I<n;I++)
{
    for (II=0;II<p;II++)
    {fscanf(fichier,"%lf %lf",&X[I][II][0],&X[I][II][1]);}
}

//CMDE 4b : calcul des ranges minimaux et maximaux
double *minrange,*maxrange;

minrange=calloc(p,sizeof(double));
maxrange=calloc(p,sizeof(double));

for (I=0;I<p;I++)
{
    minrange[I]=X[0][I][1]-X[0][I][0];
    maxrange[I]=X[0][I][1]-X[0][I][0];
    for (II=1;II<n;II++)
    {
        if (minrange[I]>(X[II][I][1]-X[II][I][0]))
        {minrange[I]=X[II][I][1]-X[II][I][0];}
        if (maxrange[I]<(X[II][I][1]-X[II][I][0]))
        {maxrange[I]=X[II][I][1]-X[II][I][0];}
    }
}

//CMDE 5 : lecture des Rš sur l'input
double *rcarre;

rcarre=calloc(K,sizeof(double));

for (I=0;I<K;I++)
{fscanf(fichier,"%lf",&rcarre[I]);}

//fermeture du fichier d'input
fclose(fichier);
free(fichier);

/***************************/
/* BLOC II : LA BOUCLE GAP */
/***************************/
```

```
125     //CMDE 6 : stockage de l'inertie totale du jeu de données original
        double **centrog;

        centrog=calloc(p,sizeof(double*));
        for (I=0;I<p;I++)
130     {centrog[I]=calloc(2,sizeof(double));}

        for (I=0;I<p;I++)
        {
            for (II=0;II<2;II++)
135         {centrog[I][II]=0;}

            for (II=0;II<n;II++)
            {
                for (III=0;III<2;III++)
140             {centrog[I][III]=centrog[I][III]+(X[II][I][III]/n);}
            }
        }

        double inertietotale;
145
        inertietotale=0;

        for (I=0;I<n;I++)
        {inertietotale=inertietotale+dist(X[I],centrog,p,dissim);}
150
        //CMDE 7 : calcul des logarithmes des inerties intra-classes
        //du jeu de données original
        double *orig;

155     orig=calloc(K,sizeof(double));

        for (I=0;I<K;I++)
        {orig[I]=log(inertietotale*(1-rcarre[I]));} //car W = T * (1-Rš)

160     //calcul des bornes de la bounding box du jeu de données original
        double *min,*max;

        min=calloc(p,sizeof(double));
        max=calloc(p,sizeof(double));
165
        for (I=0;I<p;I++)
        {
             min[I]=X[0][I][0];
             max[I]=X[0][I][1];
170          for (II=0;II<n;II++)
             {
                  if (min[I]>X[II][I][0])
                  {min[I]=X[II][I][0];}
                  if (max[I]<X[II][I][1])
175               {max[I]=X[II][I][1];}
             }
        }

        //génération d'un entier aléatoire rendue possible
180     srand (time(NULL));

        //CMDE 8 : génération dans la bounding box d'un jeu de données uniforme
        double ****poisson;

185     poisson=calloc(B,sizeof(double***));
        for (I=0;I<B;I++)
        {
             poisson[I]=calloc(n,sizeof(double**));
             for (II=0;II<n;II++)
190          {
                 poisson[I][II]=calloc(p,sizeof(double*));
                 for (III=0;III<p;III++)
```

```
            {poisson[I][II][III]=calloc(2,sizeof(double));}
        }
}

//déclaration d'un double temporaire
double distance;

for (I=0;I<B;I++)
{
    for (II=0;II<n;II++)
    {
        for (III=0;III<p;III++)
        {
          poisson[I][II][III][0]=rand();
          poisson[I][II][III][0]=poisson[I][II][III][0]/RAND_MAX;
          poisson[I][II][III][0]=poisson[I][II][III][0]*(max[III]-min[III]-maxrange[III]);
          poisson[I][II][III][0]=poisson[I][II][III][0]+min[III]+maxrange[III]/2;

          distance=rand();
          distance=distance/RAND_MAX;
          distance=distance*(maxrange[III]-minrange[III]);
          distance=distance+minrange[III];

          poisson[I][II][III][0]=poisson[I][II][III][0]-distance/2;
          poisson[I][II][III][1]=poisson[I][II][III][0]+distance;
        }
    }
}

//CMDE 9 : calcul des logarithmes des inerties intra-classes des jeux de
//données uniformes
double *artif,**tempw;

artif=calloc(K,sizeof(double));

tempw=calloc(B,sizeof(double*));
for (I=0;I<B;I++)
{tempw[I]=calloc(K,sizeof(double));}

for (I=0;I<K;I++)
{artif[I]=0;}

for (I=0;I<B;I++)
{
    tempw[I]=intraI(poisson[I],K,n,p,dissim);

    for (II=0;II<K;II++)
    {artif[II]=artif[II]+((tempw[I][II])/B);}
}

//CMDE 10 : calcul du vecteur "GAP"
double *gap;

gap=calloc(K,sizeof(double));

for (I=0;I<K;I++)
{gap[I]=artif[I]-orig[I];}

//CMDE 11 : calcul des écarts-types des logarithmes des
//inerties intra-classes des jeux de données uniformes
double *sd;

sd=calloc(K,sizeof(double));

for (I=0;I<K;I++)
{
    sd[I]=0;
    for (II=0;II<B;II++)
```

```
            {sd[I]=sd[I]+((tempw[II][I]-artif[I])*(tempw[II][I]-artif[I])/B);}
            sd[I]=sqrt(sd[I]);
        }

        //CMDE 12 : BOUCLES GAP
        for (I=0;I<K;I++)
        {
            printf("artif[%d]=%lf\n",I+1,artif[I]);
            printf("orig[%d]=%lf\n\n",I+1,orig[I]);
        }
        printf("\n");

        for (I=0;I<K;I++)
        {printf ("gap[%d]=%lf\n",I+1,gap[I]);}
        printf("\n");

        for (I=0;I<K;I++)
        {printf("sd[%d]=%lf\n",I+1,sd[I]);}

        //---output de l'exécution pure---
        I=0;
        while (gap[I]<gap[I+1]-(sd[I+1]*W))
        {I++;}
        if (gap[I]>=gap[I+1]-(sd[I+1]*W))
        {
           printf("\n\nNombre de classes naturelles sorti ");
           printf("par l'algorithme :\nk* = %d\n\n",I+1);
        }

        //désallocation générale de mémoire
        for (I=0;I<n;I++)
        {
            for (II=0;II<p;II++)
            {free (X[I][II]);}
            free (X[I]);
        }
        free (X);

        for (I=0;I<p;I++)
        {free (centrog[I]);}
        free (centrog);

        free (rcarre);

        free (orig);

        free (min);

        free (max);

        for (I=0;I<B;I++)
        {
            for (II=0;II<n;II++)
            {
                for (III=0;III<p;III++)
                {free (poisson[I][II][III]);}
                free (poisson[I][II]);
            }
            free (poisson[I]);
        }
        free (poisson);

        free (artif);

        free (gap);

        free (sd);
```

```
        for (I=0;I<B;I++)
330     {free (tempw[I]);}
        free (tempw);

        //arrêt du programme
        printf("\n\n\n");
335     system ("PAUSE");
        return (0);
    }

    /****************************/
340 /* BLOC III : SOUS-ROUTINES */
    /****************************/

    /***** CMDE i3 *****/
    //CALCUL DE L'INERTIE INTRA-CLASSES d'un jeu de données Y partitionné (de
345 //taille nn et de dimension pp), en 1 <= kk <= KK classes, sur base de la méthode
    //des hyperrectangles de gravité mobiles

    //Cette méthode est conditionnée par la dissimilarité "diss"
    //(L1, L2, ou Hausdorff)
350
    double *intraI(double ***Y,int KK,int nn,int pp,int diss)
    {
          int haz,haz2;
          int h,i,j,k;
355       int test,place;
          int *card,*aff;

          card=calloc(KK,sizeof(int));

360       aff=calloc(nn,sizeof(int));

          double distancemin;
          double ***centre,**prak;

365       centre=calloc(KK,sizeof(double**));
          for (k=0;k<KK;k++)
          {
              centre[k]=calloc(pp,sizeof(double*));
              for (j=0;j<pp;j++)
370           {centre[k][j]=calloc(2,sizeof(double));}
          }

          prak=calloc(nn,sizeof(double*));
          for (i=0;i<nn;i++)
375       {prak[i]=calloc(KK,sizeof(double));}

          double *rez;
          rez=calloc(KK,sizeof(double));

380       //initialisation des KK centres intervalles DIFFERENTS
          for (k=0;k<KK;k++)
          {
             if (k==0)
             {haz=(rand() % nn);}
385          else
             {
                do
                {haz2=(rand() % nn);}
                while (haz2==haz);
390             haz=haz2;
             }
             for (j=0;j<pp;j++)
             {
                 centre[k][j][0]=Y[haz][j][0];
395              centre[k][j][1]=Y[haz][j][1];
             }
```

```
        }

        for (h=0;h<KK;h++)
400     {
            //initialisation de la table d'affectation
            for (i=0;i<nn;i++)
            {aff[i]=0;}

405         //exécution de la méthode des hyperrectangles de gravité mobiles
            test=1;
            while (test==1)
            {
                test=0;
410
                //calcul des distances au carré séparant les objets
                //des centres intervalles actuels
                for (i=0;i<nn;i++)
                {
415                 for (k=0;k<=h;k++)
                    {prak[i][k]=dist(Y[i],centre[k],pp,diss);}
                }

                //réinitialisation des cardinaux des classes potentielles
420             for (k=0;k<=h;k++)
                {card[k]=0;}

                //affectation sur base des distances
                //séparant objets et centres intervalles
425             for (i=0;i<nn;i++)
                {
                    distancemin=prak[i][0];
                    place=0;
                    for (k=0;k<=h;k++)
430                 {
                        if (distancemin>prak[i][k])
                        {
                           distancemin=prak[i][k];
                           place=k;
435                     }
                    }
                    if (place!=aff[i])
                    {test=1;}

440                 aff[i]=place;
                    card[aff[i]]++;
                }

                //mise à jour des centres intervalles
445             for (k=0;k<=h;k++)
                {
                   for (j=0;j<pp;j++)
                   {
                       centre[k][j][0]=0;
450                    centre[k][j][1]=0;
                   }
                }
                for (i=0;i<nn;i++)
                {
455                for (j=0;j<pp;j++)
                   {
                       centre[aff[i]][j][0]=centre[aff[i]][j][0]
                       +(Y[i][j][0]/card[aff[i]]);
                       centre[aff[i]][j][1]=centre[aff[i]][j][1]
460                    +(Y[i][j][1]/card[aff[i]]);}
                }
            }

            //calcul de l'inertie intra-classe résultante
```

```
465             rez[h]=0;
                for (i=0;i<nn;i++)
                {rez[h]=rez[h]+dist(Y[i],centre[aff[i]],pp,diss);}
                rez[h]=log(rez[h]);
        }
470
        return rez;

        //désallocation de mémoire interne à la procédure
        free (card);
475
        free (aff);

        for (i=0;i<KK;i++)
        {
480         for (j=0;j<pp;j++)
            {free (centre[i][j]);}
            free (centre[i]);
        }
        free (centre);
485
        for (i=0;i<nn;i++)
        {free (prak[i]);}
        free (prak);

490     free (rez);
}

//procédure calculant une distance au carré entre deux vecteurs
//sur base de la dissimilarité "diss"
495 double dist(double **arg1,double **arg2,int pp,int diss)
{
        double rez;

        int j;
500
        rez=0;

        if (diss==1)
        {
505        for (j=0;j<pp;j++)
           {rez=rez+(fabs(arg1[j][0]-arg2[j][0])+fabs(arg1[j][1]-arg2[j][1]));}
        }
        if (diss==2)
        {
510        for (j=0;j<pp;j++)
           {rez=rez+((arg1[j][0]-arg2[j][0])*(arg1[j][0]-arg2[j][0])+(arg1[j][1]-arg2[j][1])*(arg1[j][1]-arg2[j]
        }
        if (diss==3)
        {
515        for (j=0;j<pp;j++)
           {rez=rez+maximum(fabs(arg1[j][0]-arg2[j][0]),fabs(arg1[j][1]-arg2[j][1]));}
        }

        return rez;
520 }

//procédure sélectionnant le maximum de deux arguments
double maximum(double arga,double argb)
{
525     double rez;

        if (arga>=argb)
        {rez=arga;}
        else
530     {rez=argb;}

        return rez;}
```

A.2.2 Implémentation la méthode dans le cas symbolique intervalle par modélisation ML

La méthode ici se base sur une particularisation du jeu de données symbolique intervalle par une modélisation milieu/longueur.

Le praticien a le choix en ce qui concerne la sélection de la variable la plus discriminante Y_d (commande 5c) :

- il peut soit sélectionner cette variable sur base des mesures de discrimination $cor_j k$, enregistrés par la commande 5c,

- il peut aussi faire fi de cette commande, pour entrer manuellement l'indice de la variable selon laquelle il désire exécuter sa représentation ML (ligne 134 du code).

L'output du programme est de la forme suivante.

```
RESULTATS des V = 10000 executions M/L

k* = 1 ---> 100.000000 pcnts
k* = 2 ---> 0.000000 pcnts
k* = 3 ---> 0.000000 pcnts
k* = 4 ---> 0.000000 pcnts
k* = 5 ---> 0.000000 pcnts
k* = 6 ---> 0.000000 pcnts
k* = 7 ---> 0.000000 pcnts
k* = 8 ---> 0.000000 pcnts
k* = 9 ---> 0.000000 pcnts
k* = 10 ---> 0.000000 pcnts
```

FIG. A.2 – Output de la méthode sur un jeu de données quelconque

Le programme effectue V classifications, en $k = 1, \ldots, K$ classes, du jeu de données bidimensionnel qu'est la représentation milieu/longueur. Sur chacune de ses partitions en $k = 1, \ldots, K$ classes, la règle d'arrêt de la méthode décrite au troisième chapitre est appliquée. A chaque nombre de classes est ainsi associé un pourcentage de partitions pour lesquelles le nombre de classes en question a été décidé.

Dans l'output (figure A.2), $V = 10000$.

Header du code principal : *gap3.h*

```
/* HEADER ATTACHE AU PROGRAMME gap3.c */

double *intraI(double **Y,int KK,int nn,int pp);

double dist(double *arg1,double *arg2,int pp);
```

Listing du code principal : *gap3.c*

```
/* DETERMINATION DU NOMBRE DE CLASSES AU MOYEN DE LA GAP STATISTIC
(CAS SYMBOLIQUE-INTERVALLE), modélisation milieu-longueur */

/*** Auteur : S. Cosme, 2007, F. U. N. D. P.
PROMOTEUR DE MEMOIRE : A. Hardy ***/

#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include "gap3.h" //header attaché

main ()
{

     /*******************************/
     /* BLOC I : LECTURE DE L'INPUT */
     /*******************************/

     printf("DETERMINATION DU NOMBRE DE CLASSES PAR LA GAP STATISTIC\n");
     printf("(cas symbolique-intervalle), modélisation milieu-longueur\n");
     printf("============================================================\n\n");

     //déclaration de compteurs entiers
     int I,II,III;

     //CMDE 1 : déclaration de constantes utiles
     int B,K;

     B=25;
     K=10;

     //calcul de la constante de rejet
     double W;

     W=B;
     W=sqrt(1+(1/W));

     //CMDE 2 : ouverture du fichier "txt" input en mode lecture
     char *finput;

     finput=calloc(30,sizeof(char));

     printf("FICHIER TXT D'INPUT : ");
     scanf("%s",finput);

     FILE *fichier=NULL;
     fichier=fopen(finput,"r");

     free(finput);

     //CMDE 3 : lecture des taille et dimension du jeu de données original
     int n,p;

     fscanf(fichier,"%d %d",&n,&p);

     //réduction du nombre de classes maximal à un entier inférieur à la taille
     //du jeu de données original si nécessaire
```

```
        if (K>=n)
60      {K=n-1;}

        //CMDE 4 : lecture de la matrice des données classiques du jeu de données
        //original
        double ***X;
65
        X=calloc(n,sizeof(double**));
        for (I=0;I<n;I++)
        {
            X[I]=calloc(p,sizeof(double*));
70          for (II=0;II<p;II++)
            {X[I][II]=calloc(2,sizeof(double));}
        }

        for (I=0;I<n;I++)
75      {
            for (II=0;II<p;II++)
            {fscanf(fichier,"%lf %lf",&X[I][II][0],&X[I][II][1]);}
        }

80      //CMDE 5 : lecture des Rš sur l'input
        double *rcarre;

        rcarre=calloc(K,sizeof(double));

85      for (I=0;I<K;I++)
        {fscanf(fichier,"%lf",&rcarre[I]);}

        //CMDE 5b : lecture des paramètres de discrimination
        double **cor;
90
        cor=calloc(p,sizeof(double*));
        for (I=0;I<p;I++)
        {cor[I]=calloc(K-1,sizeof(double));}

95      for (I=0;I<p;I++)
        {
            for (II=0;II<K-1;II++)
            {fscanf(fichier,"%lf",&cor[I][II]);}
        }
100
        //fermeture du fichier d'input
        fclose(fichier);
        free(fichier);

105     /*****************************/
        /* BLOC II : LA BOUCLE GAP */
        /*****************************/

        //CMDE 5c : SELECTION DE LA VARIABLE SELON LAQUELLE MODELISER PAR M/L
110     int ncldiscrim,vardiscrim;

        char discrim[1];

        printf("\n\nSELECTIONNER LA VARIABLE LA PLUS DISCRIMINANTE\n");
115     printf("SELON MESURES DE DISCRIMINATION ? O/N\n\n");
        scanf("%s",discrim);

        //UTILISATION DES MESURES DE DISCRIMINATION EN INPUT ?
        if ((strcmp(discrim,"o")==0) || (strcmp(discrim,"O")==0)) //oui
120     {
           printf("\n\nINTRODUIRE LE NOMBRE DE CLASSES\n");
           printf("POUR LEQUEL CHOISIR LA MESURE\n");
           printf("DE DISCRIMINATION MAXIMALE : ");

125        scanf("%d",&ncldiscrim);
```

```
            vardiscrim=0;
            for (I=0;I<p;I++)
            {
130             if (cor[vardiscrim][ncldiscrim-2]<cor[I][ncldiscrim-2])
                {vardiscrim=I;}
            }
        }
        else //non
135     {
            printf("\n\nINTRODUIRE LA VARIABLE SELON LAQUELLE LA MODELISATION M/L\n");
            printf("SERA EFFECTUEE : ");
            scanf("%d",&vardiscrim);
            vardiscrim=vardiscrim-1;
140     }

        printf("\nLa variable la plus discrimante pour la représentation ML");
        printf("\nest la variable indicée : %d\n\n",vardiscrim+1);

145     //CMDE 5d : construction du jeu de données bidimensionnel M/L
        double **Z;

        Z=calloc(n,sizeof(double*));
        for (I=0;I<n;I++)
150     {Z[I]=calloc(2,sizeof(double));}

        for (I=0;I<n;I++)
        {
            Z[I][0]=(X[I][vardiscrim][0]+X[I][vardiscrim][1])/2; //milieu
155         Z[I][1]=(X[I][vardiscrim][1]-X[I][vardiscrim][0]);   //longueur
            printf("%lf %lf \n",Z[I][0],Z[I][1]);
        }

        //réduction du nombre de variables à 2 : MILIEU et LONGUEUR
160     p=2;

        //calcul des bornes de la bounding box du jeu de données M/L
        double *min,*max;

165     min=calloc(p,sizeof(double));
        max=calloc(p,sizeof(double));

        for (I=0;I<p;I++)
        {
170         min[I]=Z[0][I];
            max[I]=Z[0][I];

            for (II=0;II<n;II++)
            {
175             if (min[I]>Z[II][I])
                {min[I]=Z[II][I];}
                if (max[I]<Z[II][I])
                {max[I]=Z[II][I];}
            }
180     }

        //génération d'un entier aléatoire rendue possible
        srand (time(NULL));

185     //CMDE 8 : génération dans la bounding box des processus de Poisson
        double ***poisson;

        poisson=calloc(B,sizeof(double**));
        for (I=0;I<B;I++)
190     {
            poisson[I]=calloc(n,sizeof(double*));
            for (II=0;II<n;II++)
            {poisson[I][II]=calloc(p,sizeof(double));}
        }
```

```
195
        for (I=0;I<B;I++)
        {
            for (II=0;II<n;II++)
            {
200             for (III=0;III<p;III++)
                {
                  poisson[I][II][III]=rand();
                  poisson[I][II][III]=poisson[I][II][III]/RAND_MAX;
                  poisson[I][II][III]=poisson[I][II][III]*(max[III]-min[III]);
205               poisson[I][II][III]=poisson[I][II][III]+min[III];
                }
            }
        }

210     //CMDE 9 : calcul des logarithmes des inerties intra-classes des processus
        //de Poisson
        double *artif,**tempw;

        artif=calloc(K,sizeof(double));
215
        tempw=calloc(B,sizeof(double*));
        for (I=0;I<B;I++)
        {tempw[I]=calloc(K,sizeof(double));}

220     for (I=0;I<K;I++)
        {artif[I]=0;}

        for (I=0;I<B;I++)
        {
225         tempw[I]=intraI(poisson[I],K,n,p);
            for (II=0;II<K;II++)
            {artif[II]=artif[II]+((tempw[I][II])/B);}
        }

230     //CMDE 10 : calcul du vecteur "GAP"
        double *gap;

        gap=calloc(K,sizeof(double));

235     //CMDE 11 : calcul des écarts-types des logarithmes des
        //inerties intra-classes des processus de Poisson
        double *sd;

        sd=calloc(K,sizeof(double));
240
        for (I=0;I<K;I++)
        {
            sd[I]=0;
            for (II=0;II<B;II++)
245         {sd[I]=sd[I]+((tempw[II][I]-artif[I])*(tempw[II][I]-artif[I])/B);}
            sd[I]=sqrt(sd[I]);
        }

        //CMDE 12 : BOUCLE GAP
250     const int V=10000;

        int nbrtest;
        nbrtest=V;

255     double *afftest;
        afftest=calloc(K,sizeof(double));

        for (I=0;I<K;I++)
        {afftest[I]=0;}
260
        double *orig;
        orig=calloc(K,sizeof(double));
```

```
        while (nbrtest>0)
265     {
            orig=intraI(Z,K,n,p);

            for (I=0;I<K;I++)
            {gap[I]=artif[I]-orig[I];}
270
            //---output de l'exécution---
            I=0;
            while (gap[I]<gap[I+1]-(sd[I+1]*W))
            {I++;}
275         if (gap[I]>=gap[I+1]-(sd[I+1]*W))
            {afftest[I]++;}
            nbrtest--;
        }

280     //IMPRESSION DES RESULTATS
        printf("\n\nRESULTATS des V = %d executions M/L\n\n",V);

        for (I=0;I<K;I++)
        {printf("k* = %d ---> %lf pcnts\n",I+1,afftest[I]/V*100);}
285
        //désallocation générale de mémoire
        for (I=0;I<n;I++)
        {
            for (II=0;II<p;II++)
290         {free (X[I][II]);}
            free (X[I]);
        }
        free (X);

295     free (rcarre);

        free (orig);

        free (min);
300
        free (max);

        for (I=0;I<B;I++)
        {
305         for (II=0;II<n;II++)
            {free (poisson[I][II]);}
            free (poisson[I]);
        }
        free (poisson);
310
        free (artif);

        free (gap);

315     free (sd);

        for (I=0;I<B;I++)
        {free (tempw[I]);}
        free (tempw);
320
        free (discrim);

        for (I=0;I<p;I++)
        {free (cor[I]);}
325     free (cor);

        //arrêt du programme
        printf("\n\n\n");
        system ("PAUSE");
330     return (0);
```

```
}

/*****************************/
/* BLOC III : SOUS-ROUTINES */
/*****************************/

/***** CMDE 13 *****/
//CALCUL DE L'INERTIE INTRA-CLASSES d'un jeu de données Y partitionné (de
//taille nn et de dimension pp), en 1 <= kk <= KK classes, sur base de la méthode
//des centres mobiles

double *intraI(double **Y,int KK,int nn,int pp)
{
       int haz,haz2;
       int h,i,j,k;
       int test,place;
       int *card,*aff;

       card=calloc(KK,sizeof(int));

       aff=calloc(nn,sizeof(int));

       double distancemin;
       double **centre,**prak;

       centre=calloc(KK,sizeof(double*));
       for (k=0;k<KK;k++)
       {centre[k]=calloc(pp,sizeof(double));}

       prak=calloc(nn,sizeof(double*));
       for (i=0;i<nn;i++)
       {prak[i]=calloc(KK,sizeof(double));}

       double *rez;
       rez=calloc(KK,sizeof(double));

       //cas de l'inertie totale (k=1)
       rez[0]=0;
       for (i=0;i<nn;i++)
       {
           for (j=0;j<nn;j++)
           {rez[0]=rez[0]+dist(Y[i],Y[j],pp)/(2*nn);}
       }
       rez[0]=log(rez[0]);

       //initialisation des KK centres DIFFERENTS
       for (k=0;k<KK;k++)
       {
          if (k==0)
          {haz=(rand() % nn);}
          else
          {
             do
             {haz2=(rand() % nn);}
             while (haz2==haz);
             haz=haz2;
          }
          for (j=0;j<pp;j++)
          {centre[k][j]=Y[haz][j];}
       }

       for (h=1;h<KK;h++)
       {
            //initialisation de la table d'affectation
            for (i=0;i<nn;i++)
            {aff[i]=0;}

            //exécution de la méthode des centres mobiles
```

```
            test=i;
400         while (test==i)
            {
                  test=0;

                  //calcul des distances au carré séparant les objets
405               //des centres actuels
                  for (i=0;i<nn;i++)
                  {
                      for (k=0;k<=h;k++)
                      {prak[i][k]=dist(Y[i],centre[k],pp);}
410               }

                  //réinitialisation des cardinaux des classes potentielles
                  for (k=0;k<=h;k++)
                  {card[k]=0;}
415
                  //affectation sur base des distances
                  //séparant objets et centres
                  for (i=0;i<nn;i++)
                  {
420                   distancemin=prak[i][0];
                      place=0;
                      for (k=0;k<=h;k++)
                      {
                          if (distancemin>prak[i][k])
425                       {
                             distancemin=prak[i][k];
                             place=k;
                          }
                      }
430                   if (place!=aff[i])
                      {test=i;}

                      aff[i]=place;
                      card[aff[i]]++;
435               }

                  //mise à jour des centres
                  for (k=0;k<=h;k++)
                  {
440                  for (j=0;j<pp;j++)
                     {centre[k][j]=0;}
                  }
                  for (i=0;i<nn;i++)
                  {
445                  for (j=0;j<pp;j++)
                     {centre[aff[i]][j]=centre[aff[i]][j]
                     +(Y[i][j]/card[aff[i]]);}
                  }
            }
450
            //calcul de l'inertie intra-classe résultante
            rez[h]=0;
            for (i=0;i<nn;i++)
            {rez[h]=rez[h]+dist(Y[i],centre[aff[i]],pp);}
455         rez[h]=log(rez[h]);
      }

      return rez;

460   //désallocation de mémoire interne à la procédure
      free (card);
      free (aff);
      for (i=0;i<KK;i++)
      {free (centre[i]);}
465   free (centre);
      for (i=0;i<nn;i++)
```

```
        {free (prak[i]);}
        free (prak);
        free (rez);
}

//procédure calculant une distance au carré entre deux vecteurs
double dist(double *arg1,double *arg2,int pp)
{
        double rez;

        int i;

        rez=0;

        for (i=0;i<pp;i++)
        {rez=rez+(arg1[i]-arg2[i])*(arg1[i]-arg2[i]);}

        return rez;
}
```

Bibliographie

[1] ANDERBERG M. R. (1973), *Cluster Analysis for Applications*, Academic Press, New-York.

[2] ANDERSON E. (1935), *The irises of the Gaspe Peninsula*, Bulletin of the American Iris Society, 59, 2-5.

[3] BAKER F. B., HUBERT L. J. (1975), *Measuring the power of hierarchical cluster analysis*, Journal of the American Statistical Association, 70, 31-38.

[4] BEALE E. M. L. (1969), *Cluster Analysis*, London : Scientific Control Systems.

[5] BERTRAND P., BEL MUFTI G. (2006), *Loevinger's measures of rule quality for assessing cluster stability*, Computational Statistics and Data Analysis, 50, 992-1015.

[6] BOCK H.H., DIDAY E. (2000), *Analysis of symbolic data. Exploratory methods for extracting statistical information from complex data*, Springer-Verlag.

[7] CALINSKI R. B., HARABASZ J. (1974), *A dendrite method for cluster analysis*, Communications in Statistics, 3, 1-27.

[8] CHANDON J. L., PINSON S. (1981), *Analyse typologique : théorie et applications*, Masson, Paris.

[9] DUDA R. O., HART P. E. (1973), *Pattern classification and scene analysis*, Wiley-Interscience, New York.

[10] EVERITT B. (1993), *Cluster Analysis*, Arnold, London.

[11] GORDON, A. D. (1996), *Cluster Validation*, Data Science, Classification, and Related Methods, Springer, 22-39.

[12] HARDY A., *Aspects statistiques de la classification*, cours, FUNDP - Université de Namur, Namur.

[13] HARDY A. (1996), *On the number of clusters*, Computational Statistics and Data Analysis, 23, 83-96.

[14] HARDY A. (2005), *Validation in unsupervised symbolic classification*, Proceedings du Congrès ASMDA, Brest.

[15] HARDY A., BAUNE J. (2006), *Clustering and validation of interval data*, Department of Mathematics, FUNDP - Université de Namur, Namur.

[16] HUBERT L. J., LEVIN J. R. (1976), *A general statistical framework for assessing categorical clustering in free recall*, Psychological Bulletin, 83, 1072-1080.

[17] JAIN A. K., DUBES R.C. (1988), *Algorithms for Clustering Data*, Prentice-Hall, Inc., Upper Saddle River, NJ.

[18] JAMBU M. (1978), *Classification Automatique pour l'Analyse des données*, North-Holland, Amsterdam.

[19] KRZANOWSKI W. J., LAI Y. T. (1985), *A criterion for determining the number of groups in a data set using sum of squares clustering*, Biometrics, 44, 23-34.

[20] MILLIGAN G. W., COOPER M. C. (1985), *An examination of procedures for determining the number of clusters in a data set*, Psychometrika, 50, 159-179.

[21] MOORE M. (1984), *On the estimation of a convex set*, The Annals of Statistics, 12, 3, 1090-1099.

[22] RIPLEY B. D., RASSON J.-P. (1977), *Finding the edge of a Poisson forest*, Journal of Applied Probability, 14, 483-491.

[23] SARLE W. S. (1983), *Cubic Clustering Criterion*, Technical Report : A-108, SAS Institute Inc..

[24] STRUYF A., HUBERT M., ROUSSEEUW P. J. (1996), *Clustering in an Objected-Oriented Environment*, Journal of Statistical Software, 1.

[25] TIBSHIRANI R., WALTHER G., HASTIE T. (2000), *Estimating the number of clusters in a dataset via the Gap Statistic*, University of Stanford, CA.

[26] VERDE R., DE CARVALHO F., LECHEVALLIER Y. (2000), *A dynamical clustering algorithm for multi-nominal data*, Data analysis, classification, and related methods, H. Kiers et al. (Eds), 387-393.

[27] Référence internet, European Jobs : www.dm.unibo.it/~simoncin/EuropeanJobs.html.

[28] Référence internet, SODAS 2 : http ://www.info.fundp.ac.be/asso.