6 PARTITIONNEMENT DE DOCUMENTS

Nous nous intéressons dans ce chapitre au partitionnement de documents, c'est-à-dire à la répartition en K classes disjointes d'une collection de documents donnée. Comme pour la majorité des chapitres précédents, les méthodes que nous allons passer en revue ne sont pas spécifiques aux documents textuels et peuvent être appliquées à toute collection d'objets, que ces objets soient des textes, des images, des vidéos, ou des logs (journaux des opérations) de machine par exemple. Outre le nom de *partitionnement*, plusieurs autres termes sont utilisés pour désigner la répartition d'un ensemble d'objets en plusieurs classes. On parle ainsi de *clustering*, qui est bien sûr un anglicisme, même si ce terme désigne à la fois le partitionnement (hard clustering ou *clustering dur*) et une répartition plus souple, en classes non nécessairement disjointes (soft clustering ou *clustering mou*). Le terme de *classification* est à préférer à celui de *clustering*, mais est ambigu car il désigne aussi parfois la catégorisation. Pour lever cette ambiguïté, certains auteurs parlent de *classification supervisée* pour désigner la catégorisation, et de *classification non supervisée* pour le *clustering*. Le partitionnement serait alors la *classification non supervisée dure*, une dénomination relativement lourde. De ce fait, dans certains travaux, le terme seul de *classification* est utilisé, le contexte permettant en général de savoir à quel type de classification l'on a affaire. Afin d'éviter toute ambiguïté, nous parlerons de *partitionnement* dans ce chapitre, la problématique de la *classification non supervisée molle* étant abordée dans le chapitre suivant.

Dans la suite de ce chapitre, nous considérons une collection $\mathcal{C}$ de N documents, et faisons l'hypothèse que chaque document est représenté par un vecteur de termes tel que défini dans le chapitre 2. Comme précédemment, $\mathbf{d}_j$ désigne la représentation vectorielle du j^e document de la collection (section 2.3, chapitre 2).

6.1 Définitions

Le partitionnement a pour but l'identification de classes disjointes de documents au sein d'une collection donnée. On cherche bien sûr, d'une part, à obtenir des classes homogènes, c'est-à-dire qui rassemblent des documents proches les uns des autres, et, d'autre part, à éviter que des classes différentes soient proches l'une de l'autre (si tel était le cas, on aurait intérêt à les fusionner). On dit souvent que l'ensemble des classes structure la collection.

Nous noterons $\mathcal{G}$ l'ensemble des partitions que l'on peut obtenir à partir d'une collection $\mathcal{C}$ de N documents. Le cardinal de $\mathcal{G}$ est donné par le nombre de Bell B_N :

$$B_{n+1} = \sum_{k=0}^{n} C_n^k B_k = \sum_{k=0}^{n} \frac{n!}{(n-k)!k!} B_k$$

avec $B_0 = B_1 = 1$ (voir l'exercice 6.1).

Dans le cas (idéal) où l'on dispose d'une fonction $\mathcal{F}$ permettant d'associer un coût à chaque partition, le problème du partitionnement consiste à rechercher, parmi toutes les partitions possibles, celle de meilleur coût :

$$\text{argmin}_{G \in \mathcal{G}} \mathcal{F}(G) \tag{6.1}$$

où G désigne une partition quelconque [1]. Cette formulation n'est toutefois que d'un intérêt limité en pratique, d'une part car l'on ne dispose pas toujours d'une fonction de coût, et d'autre part car le nombre de partitions possibles (nombre de Bell) est en général trop élevé pour qu'une recherche exhaustive soit menée. Par exemple, pour $N = 6$ il y a 203 partitions possibles, 115 975 partitions possibles pour 10 documents, et plus d'un milliard pour 15 documents. En pratique, comme nous le verrons dans la suite, seul un petit nombre de partitions est en général examiné par les algorithmes de partitionnement. Nous utiliserons dans la suite les notations et éléments suivants.

Définition 1. Classe. *Soit G une partition d'une collection $\mathcal{C}$ donnée de N documents. Un élément de G est appelé une classe. Une classe correspond donc à un sous-ensemble des documents de $\mathcal{C}$. Une classe d'une partition G sera de plus notée G_i, où $1 \leq i \leq |G|$.*

Le lecteur aura noté que cette définition est une définition *a posteriori* dans la mesure où l'on a défini une classe comme un élément d'une partition, en général obtenue par un algorithme de partitionnement. Il n'existe

1. Nous utilisons ici la notation G pour groupement, la notation P étant réservée aux probabilités.

pas de définition claire de ce qu'est une classe *a priori*, qui figurerait dans les données indépendamment de toute représentation retenue et tout algorithme de partitionnement. De même, il n'existe en général pas un nombre idéal de classes (on peut certes construire des exemples simples pour lesquels classes et nombre de classes semblent bien définis, mais ces exemples jouets sont généralement très éloignés des collections que l'on observe en pratique). C'est dire que la problématique du partitionnement, et plus généralement celle de la classification, est en partie subjective, les classes recherchées dépendant des applications et analyses visées.

Définition 2. Représentant de classe. *Soit G_i une classe d'une partition G. On appelle représentant (ou prototype) de G_i un vecteur à V dimensions $\mathbf{r}_i$ résumant la classe. Le centroïde ou centre de gravité de la classe est souvent utilisé comme représentant.*

La figure 6.1 illustre les notions ci-dessus.

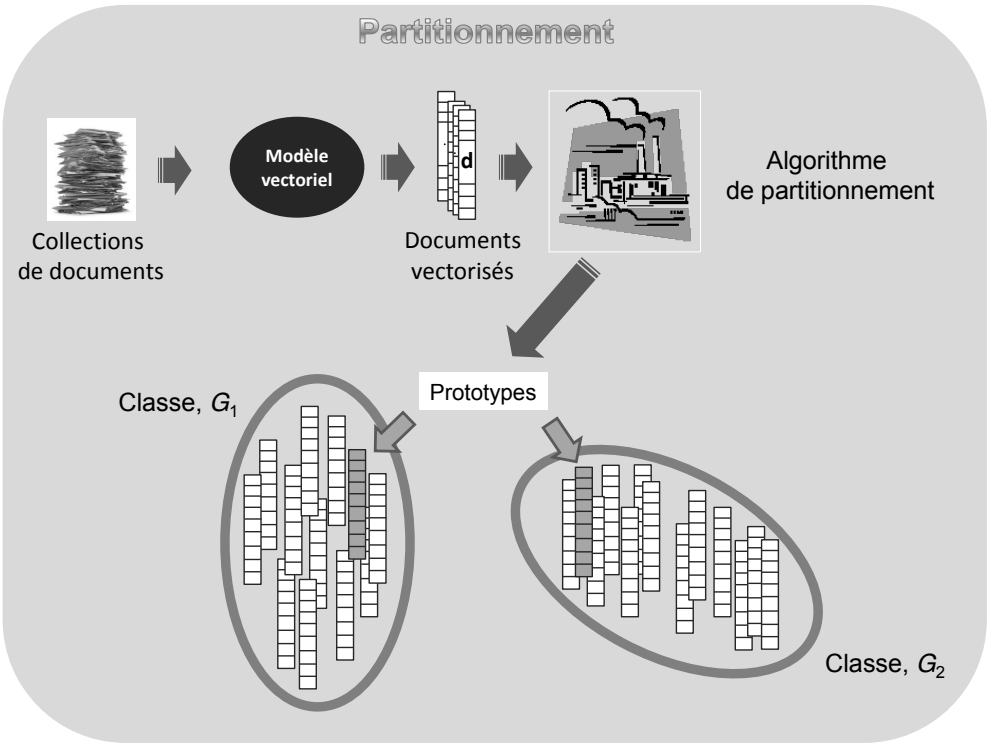

Figure 6.1 : Illustration des notions de *classes* et de *prototypes*.

6.2 Les étapes du partitionnement

Le partitionnement d'un ensemble de documents en différentes classes est en général un processus itératif au cours duquel un utilisateur cherche à mieux comprendre le contenu d'une collection, ou à en tirer des informations utiles. Ce processus passe par les étapes suivantes :

1. Choix d'une mesure de similarité et éventuellement calcul de la matrice de similarité.
2. Partitionnement.
   a. Choix de la méthode de partitionnement.
   b. Choix de l'algorithme de partitionnement.
3. Validation des classes obtenues.
4. Retour à l'étape 2, en modifiant les paramètres de l'algorithme de partitionnement voire la méthode de partitionnement et l'algorithme associé ; ce retour est bien sûr optionnel.

La première étape, choix d'une similarité et calcul de la matrice de similarité, consiste tout d'abord à sélectionner une mesure de similarité (ou de distance) appropriée aux documents retenus, puis, éventuellement, à calculer la similarité (ou distance) entre chaque couple de documents. Il existe bien sûr plusieurs mesures de similarité ou de distance, les plus connues et les plus utilisées étant l'indice de Jaccard, le coefficient de Dice, le cosinus et la distance euclidienne. Toutes ces mesures ont été et sont toujours couramment utilisées dans le cas de données textuelles, le cosinus étant certainement la mesure la plus populaire dans ce contexte. Nous allons illustrer ces différentes mesures sur un exemple simple.

Nous considérons ici une collection simplifiée dans laquelle les poids des termes dans les documents sont des poids de présence/absence binaires (1 si le terme est présent, 0 sinon). Cette collection, qui contient 4 documents (d_1, d_2, d_3, d_4) et dont le vocabulaire d'indexation comprend 5 termes (*java, langage, café, programmation, récolte*), est résumée dans la matrice documents-termes ci-dessous :

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{array}{c} \begin{array}{ccccc} \text{java} & \text{langage} & \text{café} & \text{programmation} & \text{récolte} \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Les mesures de similarité et distance mentionnées ci-dessus prennent la forme suivante :

– L'*indice de Jaccard*, sous sa forme la plus simple, calcule la proportion de termes communs à deux documents. Dans le cas de poids quelconques compris entre 0 et 1, cet indice prend la forme :

$$\text{sim}_{\text{Jaccard}}(d, d') = \frac{\sum_{i=1}^{V} w_{id} w_{id'}}{\sum_{i=1}^{V} w_{id} + w_{id'} - w_{id} w_{id'}}$$

Sur l'exemple précédent, cet indice fournit la matrice de similarité suivante :

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\ \begin{pmatrix} 1 & 0,33 & 0 & 0,25 \\ 0,33 & 1 & 0 & 0,25 \\ 0 & 0 & 1 & 0,66 \\ 0,25 & 0,25 & 0,66 & 1 \end{pmatrix} \end{array}$$

Dans la mesure où l'indice de similarité de Jaccard est symétrique en d et d', la matrice obtenue est elle-même symétrique.

– Le *coefficient de Dice* prend la forme :

$$\text{sim}_{\text{Dice}}(d, d') = \frac{\sum_{i=1}^{V} w_{id} w_{id'}}{\sum_{i=1}^{V} w_{id}^2 + w_{id'}^2}$$

Sur l'exemple précédent, ce coefficient fournit la matrice de similarité symétrique :

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\ \begin{pmatrix} 1 & 0,25 & 0 & 0,20 \\ 0,25 & 1 & 0 & 0,25 \\ 0 & 0 & 1 & 0,40 \\ 0,20 & 0,25 & 0,40 & 1 \end{pmatrix} \end{array}$$

– Le *cosinus*, lui, s'écrit :

$$\text{sim}_{\cos}(d, d') = \frac{\sum_{i=1}^{V} w_{id} w_{id'}}{\sqrt{\sum_{i=1}^{V} w_{id}^2} \sqrt{\sum_{i=1}^{V} w_{id'}^2}}$$

Sur l'exemple précédent, le cosinus fournit la matrice de similarité symétrique :

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\ \begin{pmatrix} 1 & 0,5 & 0 & \frac{1}{\sqrt{6}} \\ 0,5 & 1 & 0 & \frac{1}{\sqrt{6}} \\ 0 & 0 & 1 & \frac{\sqrt{2}}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{\sqrt{2}}{\sqrt{3}} & 1 \end{pmatrix} \end{array}$$

Tout comme l'indice de Jaccard et le coefficient de Dice, le cosinus est nul si le poids de chaque terme dans l'un ou l'autre des documents comparés est nul. Dans beaucoup de systèmes de pondération (voir chapitre 2), le poids d'un terme dans un document est nul lorsque ce terme n'apparaît pas dans le document. On voit donc que les mesures précédentes donneront une similarité nulle lorsque les deux documents n'auront aucun terme en commun.
La formule du cosinus mesure en fait le cosinus de l'angle formé par les deux vecteurs $\mathbf{d}$ et $\mathbf{d}'$. Ce cosinus est nul lorsque les deux vecteurs sont orthogonaux (donc lorsqu'ils n'ont aucun terme en commun). Il est maximal lorsque les deux vecteurs sont colinéaires, c'est-à-dire lorsque les deux documents contiennent exactement les mêmes termes dans les mêmes proportions.

– Enfin, la *distance euclidienne* est donnée par :

$$\text{dist}_{\text{eucl}}(d, d') = ||\mathbf{d} - \mathbf{d}'||_2 = \sqrt{\sum_{i=1}^{V} (w_{id} - w_{id'})^2}$$

Cette distance est ensuite soit transformée en similarité, en utilisant par exemple une valeur maximale (égale à la plus grande distance observée entre deux points ou arbitrairement fixée) à laquelle on retranche la distance entre deux documents, soit utilisée telle quelle en modifiant éventuellement l'algorithme de partitionnement si ce dernier opère sur des

matrices de similarité (la similarité la plus grande correspondant dans ce cas à la distance la plus petite). La matrice de distance euclidienne sur les données précédentes vaut (ici encore, la symétrie de la mesure garantit la symétrie de la matrice de similarité) :

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\ \begin{pmatrix} 0 & \sqrt{2} & 2 & \sqrt{3} \\ \sqrt{2} & 0 & 2 & \sqrt{3} \\ 2 & 2 & 0 & 1 \\ \sqrt{3} & \sqrt{3} & 1 & 0 \end{pmatrix} \end{array}$$

Contrairement aux mesures précédentes, la distance euclidienne fournit un score non nul (et non défini *a priori*) dans le cas où les deux documents considérés n'ont aucun terme en commun. Ceci limite l'usage strict de cette mesure en conjonction avec des algorithmes de fichier inverse (puisque l'on a besoin ici de passer en revue tous les termes des documents).

Une fois la similarité choisie, et éventuellement la matrice de similarité construite, la deuxième étape réside dans le choix de la méthode de partitionnement. Ce choix dépend essentiellement des données et du type de partitionnement que l'on souhaite obtenir. On distingue en particulier deux grands types de partitionnement :

1. *Partitionnement à plat*, dans lequel les classes sont indépendantes les unes des autres. L'utilisateur choisit alors en général un nombre de classes ou un seuil sur une mesure de similarité.
2. *Partitionnement hiérarchique*, dans lequel les classes sont structurées sous forme d'une hiérarchie, en général représentée par un arbre, souvent binaire (chaque parent a deux enfants). La relation parent-enfant est interprétée en termes de généricité/spécificité, ou de niveaux d'abstraction.

L'étape suivante consiste enfin à choisir l'algorithme de partitionnement le mieux adapté à la méthode retenue.

Comme nous l'avons souligné plus haut, la recherche d'une partition utile pour une collection de documents repose en général sur un processus itératif qui donne à l'utilisateur la possibilité d'affiner ses choix (comme le nombre de classes recherchées) et lui permet de se faire une idée plus précise du contenu de la collection de documents. C'est l'étape de validation du processus global. Cette étape s'effectue parfois en contexte, lorsque le partitionnement est plongé dans une application plus large. La validation des classes est alors souvent réalisée à travers la validation de l'application plus large. Un exemple

de telle validation se trouve dans les applications visant à partitionner les résultats d'un système de recherche d'information [2] ; la validité des classes peut être mesurée dans ce cas par le temps gagné, en utilisant la partition, dans le processus de recherche d'information. En dehors d'un contexte plus large, la validation nécessite de disposer des outils permettant de visualiser les résultats du partitionnement (étape 3 du processus général décrit plus haut). Il existe plusieurs outils de visualisation de partitions. Nous renvoyons les lecteurs au chapitre 8 (Mothe (2011)) du livre *Modèles statistiques pour l'accès à l'information textuelle* (Gaussier et Yvon (2011)) pour une présentation détaillée de tels outils.

Nous allons maintenant nous intéresser aux algorithmes standard de partitionnement.

6.3 Principaux algorithmes de partitionnement

Nous allons présenter quelques-uns des principaux algorithmes de partitionnement. Le nombre d'algorithmes proposés pour cette tâche est en fait très important, et nous nous concentrerons ici sur les algorithmes les plus classiques qui comptent parmi les plus utilisés.

6.3.1 Partitionnement à plat : méthodes de réallocation

Les méthodes de (ré)allocation fonctionnent suivant le schéma simple suivant : (1) identifier des représentants de classes puis (2) affecter chaque document à la classe du représentant le plus proche, ces étapes étant éventuellement répétées. Les méthodes de réallocation ne nécessitent pas la matrice de similarité complète, mais reposent bien sûr sur une mesure de similarité (ou de distance).

Méthode à une passe

La méthode la plus simple, appelée *méthode à une passe* (en anglais *single pass method*), est décrite dans l'algorithme 16, présenté ici avec une mesure de similarité ; ce même algorithme peut bien sûr être utilisé avec une distance (en modifiant l'inégalité par rapport à la distance seuil et les max en min).

$|G_l|$ désigne le cardinal de la l^e classe de G. La mise à jour des représentants lorsqu'un nouveau document a été ajouté à une classe consiste ici à calculer le vecteur moyen des documents de la classe. Cette mise à jour

2. http://search.yippy.com/

ALGORITHME 16. Algorithme de la méthode à une passe

Entrée :
– $\mathcal{C} = \{d_1, \cdots, d_N\}$, collection de documents ;
– S_T, seuil de similarité;
Initialisation :
– $L \leftarrow 1$;
– $G_L = \{d_1\}$;
– $\mathbf{r}_L \leftarrow d_1$;
– $k \leftarrow 2$;
tant que $k \leq N$ **faire**
  $S_{\text{max}} = \max_{1 \leq l \leq L} \text{sim}(\mathbf{r}_l, d_k)$;
  $lm = \text{argmax}_{1 \leq l \leq L} \text{sim}(\mathbf{r}_l, d_k)$;
  si $S_{max} \geq S_T$ **alors**
    $G_{lm} = G_{lm} \cup \{d_k\}$;
    $\mathbf{r}_{lm} = \frac{1}{|G_{lm}|} \sum_{d \in G_{lm}} \mathbf{d}$;
  sinon
    $L \leftarrow L + 1$;
    $G_L = \{d_k\}$;
    $\mathbf{r}_L \leftarrow d_k$;
  fin
  $k \leftarrow k + 1$;
fin
Sortie : G, une partition de $\mathcal{C}$

peut aussi être effectuée différemment, par exemple en choisissant un document au hasard dans la classe ou en pondérant plus fortement les documents récemment affectés à la classe dans le cas d'une version en ligne de cet algorithme. Enfin, le seuil S_T est choisi par l'utilisateur, choix qui peut s'avérer difficile en pratique. Il est en fait souvent nécessaire de jouer avec ce genre d'algorithmes et d'effectuer des essais avec plusieurs valeurs de seuil.

La complexité de l'algorithme à une passe est de l'ordre de $O(NK)$ où K est le nombre de classes obtenues, ce nombre étant égal à N dans le pire cas, qui peut être observé si le seuil retenu S_T est trop grand. La mise en œuvre informatique de cet algorithme ne pose pas de problèmes particuliers. À noter toutefois que le choix des centroïdes pour les représentants des classes a tendance à fournir des vecteurs pleins ; l'utilisation de représentations creuses reste intéressante pour des gains d'espace, mais pas nécessairement pour des gains de temps.

Illustration Si l'on applique cet algorithme à l'exemple utilisé précédemment, en prenant pour similarité le coefficient de Jaccard, pour seuil $0,2$, et en considérant les documents dans l'ordre d_1, d_2, d_3, d_4, nous obtenons la partition suivante : $G = \{\{d_1, d_2\}, \{d_3, d_4\}\}$. Si toutefois, avec les mêmes éléments, nous considérons les documents dans l'ordre d_4, d_1, d_2, d_3, nous obtenons : $G = \{\{d_4, d_1, d_2\}, \{d_3\}\}$. Comme on le voit, un des défauts majeurs de cet algorithme est que la partition obtenue dépend de l'ordre dans lequel les documents sont considérés. Suivant cet ordre, on pourra se retrouver avec peu de très grandes classes, et un grand nombre de classes très petites, ne contenant parfois qu'un seul document. Ce défaut de la méthode à une passe est corrigé dans l'algorithme des k-moyennes (en anglais *k-means*), certainement l'un des algorithmes les plus populaires de partitionnement. Cet algorithme est aussi parfois appelé algorithme de Lloyd [3]. L'algorithme des nuées dynamiques généralise l'algorithme des k-moyennes en considérant différents types de mise à jour des représentants des classes (l'algorithme standard est fondé sur le calcul du centroïde, ou vecteur moyen, tout comme la version de l'algorithme à une passe donnée ci-dessus).

Algorithme des k-moyennes

Dans sa forme la plus standard, l'algorithme des k-moyennes vise à trouver la partition qui minimise la distance intraclasses :

$$\text{argmin}_G \underbrace{\left(\sum_{k=1}^{K} \sum_{d \in G_k} ||\mathbf{d} - \mathbf{r}_k||_2^2 \right)}_{\mathcal{L}(G_1,\ldots,G_K;\mathbf{r}_1,\ldots,\mathbf{r}_K)} \tag{6.2}$$

où K est le nombre de classes, $\mathbf{r}_k$ est le vecteur moyen des documents de G_k et $||.||_2^2$ représente le carré de la distance euclidienne. Pour simplifier la présentation, nous noterons dans la suite cette distance intraclasses par $SSR(K)$. La notation $SSR(K)$ provient de l'anglais *Sum of Squared Residuals* (somme des carrés des résidus), et l'utilisation de K comme paramètre de la fonction SSR rend compte du fait que cette somme dépend du nombre de classes retenu.

L'algorithme standard associé consiste alors, à partir d'un ensemble de représentants initiaux, à itérer les deux opérations suivantes :

– réaffecter chaque document à la classe du représentant le plus proche (étape de réaffectation) ;

3. La première forme de cet algorithme serait due à Stuart Lloyd, en 1957.

ALGORITHME 17. Algorithme des k-moyennes

Entrée :
– $\mathcal{C} = \{d_1, \cdots, d_N\}$, collection de documents ;
– K, nombre de classes ;
– T, nombre d'itérations maximal admis;
Initialisation :
– $(\mathbf{r}_1^{(0)}, \cdots, \mathbf{r}_K^{(0)})$, ensemble de représentants initiaux;
– $t \leftarrow 1$;
tant que *($t < T$) ou (l'ensemble des classes n'est pas stable)* **faire**
  pour chaque $d \in \mathcal{C}$ **faire**
    // Etape de réaffectation
    $G_k^{(t)} \leftarrow \{d : ||\mathbf{d} - \mathbf{r}_k^{(t-1)}||_2^2 \leq ||\mathbf{d} - \mathbf{r}_l^{(t-1)}||_2^2, \forall l \neq k, 1 \leq l \leq K\}$;
  fin
  pour chaque $k, 1 \leq k \leq K$ **faire**
    // Etape de recalcul des centroïdes
    $\mathbf{r}_k^{(t)} \leftarrow \frac{1}{|G_k^{(t)}|} \sum_{d \in G_k^{(t)}} \mathbf{d}$;
  fin
  $t \leftarrow t + 1$;
fin
Sortie : G, une partition de $\mathcal{C}$ en K classes

– mettre à jour les représentants en fonction des documents présents dans la classe (étape de recalcul des centroïdes).

Ces deux étapes sont explicitées dans l'algorithme 17 ci-dessous, qui prend en entrée une collection de documents, un nombre de classes et un nombre maximal d'itérations. Cet algorithme correspond à la forme standard de l'algorithme des k-moyennes.

Cet algorithme appelle un certain nombre de remarques :

1. Choix des représentants initiaux : le choix des représentants initiaux peut être réalisé de différentes façons, les deux plus populaires étant un tirage aléatoire sans remise de K documents de la collection, ou l'utilisation d'une méthode de partitionnement simple, comme la méthode à une passe. Dans ce dernier cas, il peut s'avérer nécessaire de compléter les centroïdes obtenus par d'autres représentants, tirés au hasard dans la collection, si le nombre K de classes retenu est supérieur au nombre de classes fournies par la méthode à une passe. Il est toutefois important de noter que la solution finale dépend des représentants initiaux choisis, et qu'un choix aléatoire de ces représentants rend l'algorithme non

déterministe. Pour remédier à ce problème, on exécute en pratique plusieurs fois l'algorithme des k-moyennes [4] et on choisit la partition qui minimise la distance intraclasse (équation 6.2) ;

2. Condition d'arrêt et convergence : pour assurer la convergence de l'algorithme, il est nécessaire, dans l'étape de réaffectation et en cas d'égalité de distance entre la classe courante et une autre classe, de ne pas bouger le document. On peut alors montrer (voir exercice 6.2) qu'à chaque étape de réaffectation, la distance intraclasse ($SSR(K)$) ; le recalcul des centroïdes garantit également une diminution (ou une stabilité) de la distance intraclasse (exercice 6.2). Ainsi, l'algorithme des k-moyennes converge puisqu'à chaque itération la distance intraclasse diminue. Il n'y a toutefois aucune garantie que la partition obtenue corresponde à un minimum global de la distance intraclasse ; en général, seul un minimum local de cette distance est atteint. La condition d'arrêt porte à la fois sur le nombre d'itérations et la stabilité des classes obtenues. Il est parfois difficile, à cause des problèmes d'arrondi informatique, d'imposer que la partition obtenue soit stable (la stabilité est en général testée non pas au niveau des centroïdes des classes, mais des documents constituant la classe, de façon à réduire les problèmes d'arrondi). La condition sur le nombre maximal d'itérations à réaliser permet de s'affranchir de ce problème d'arrondi. En pratique, quelques dizaines d'itérations suffisent souvent.
3. Complexité : à chaque itération, il est nécessaire de comparer les N documents aux K représentants des classes, ce qui est réalisé en $O(NK)$ (nous négligeons ici la complexité du calcul d'une distance ou d'une similarité entre deux vecteurs de taille M, qui est en $O(M)$) ; la mise à jour des représentants des classes est dominée par $O(N)$, ce qui conduit finalement à une complexité totale de l'ordre de $O(NKT)$.
4. Nombre de partitions à K classes : une des caractéristiques de cet algorithme est de fixer le nombre de classes K. Le nombre de partitions en K classes d'un ensemble à N éléments, $P_{N,K}$, est bien sûr inférieur au nombre de Bell, c'est-à-dire au nombre de partitions d'un ensemble à N éléments, mais reste encore trop grand pour une recherche exhaustive. Les nombres $P_{N,K}$, appelés *nombres de Stirling de deuxième espèce*, se calculent par récurrence, à partir de la formule suivante (voir Saporta (2011) et exercice 6.3) :

$$P_{N,K} = P_{N-1,K-1} + KP_{N-1,K}$$

4. Il faut bien sûr s'assurer que le choix des représentants initiaux est bien aléatoire ; attention à ne pas utiliser systématiquement la même graine pour la fonction *rand()* disponible dans la majorité des langages de programmation.

avec $P_{N,1} = P_{N,N} = 1$.

5. Extension hiérarchique : il est bien sûr possible d'étendre l'algorithme des k-moyennes de façon à obtenir une classification hiérarchique ; il suffit pour cela de repartitionner les classes obtenues à chaque application de l'algorithme des k-moyennes. Il est à noter que cette extension hiérarchique présente le défaut de se reposer entièrement sur l'utilisateur pour le choix de la structure hiérarchique.
6. Algorithmes des k-moyennes et EM : il existe une version *continue* de l'algorithme des k-moyennes (Bottou et Bengio (1994)) qui repose sur la réécriture de la distance intraclasses (équation 6.2) sous forme d'erreur de quantisation :

$$\text{argmin}_{\mathbf{r}} \left(\sum_{d} \frac{1}{2} \min_{k} ||\mathbf{d} - \mathbf{r}_k||_2^2 \right) \tag{6.3}$$

   Comme on le voit, sous cette formulation, équivalente à la précédente, on recherche les k représentants, $\mathbf{r}_k$, qui minimisent la distance totale aux documents. Il est possible de déployer, à partir de cette formulation, un algorithme de descente de gradient, dont la règle de mise à jour est donnée par (cf. Bottou et Bengio (1994)) :

$$\Delta \mathbf{r}_k^{(t)} = \sum_{d} \begin{cases} \epsilon_t(\mathbf{d} - \mathbf{r}_k) & \text{si } \mathbf{r}_k = \min_{k'} ||\mathbf{d} - \mathbf{r}_{k'}||_2^2 \\ 0 & \text{sinon} \end{cases} \tag{6.4}$$

   où ϵ_t correspond au taux d'apprentissage. Une telle règle de mise à jour converge dès lors que $\sum \epsilon_t = \infty$ et $\sum \epsilon_t^2 < \infty$. Un choix standard réalisant ces conditions (cf. Kohonen (1989)) est $\epsilon_t = \frac{\epsilon_0}{t}$, où ϵ_0 est une valeur par défaut choisie arbitrairement (dans beaucoup d'implantations ϵ_0 est pris égal à 1). Enfin, il est à noter que l'algorithme de descente du gradient fondé sur la règle de mise à jour ci-dessus est similaire à un algorithme EM (*Expectation-Maximization*), que nous présentons à l'exercice 6.4.
7. Choix de K : la fonction $SSR(K)$ est en fait une fonction positive décroissante en K et $SSR(N) = 0$ (voir exercice 6.2). La meilleure partition, au sens de la distance intraclasse, est donc celle obtenue avec N classes, une pour chaque document. Cette partition n'est bien sûr pas intéressante car elle n'apporte aucune information nouvelle sur la collection. De plus, la complexité du modèle de partitionnement utilisé est maximale puisqu'on fait appel au plus grand nombre de classes possible pour rendre compte de la collection. Il est possible de moduler distance

intraclasse et complexité du modèle de partitionnement sous la forme : $SSR(K) + \lambda K$, où λ est un facteur positif ou nul qui régule l'importance donnée aux deux aspects, distance intraclasse et complexité. On cherche ensuite le nombre de classes K qui minimise ce critère :

$$K = \text{argmin}_{K'}(SSR(K') + \lambda K') \tag{6.5}$$

Lorsque λ est grand, les modèles complexes (c'est-à-dire reposant sur un grand nombre de classes) sont pénalisés ; en revanche, lorsque λ est faible, c'est la distance intraclasse qui est privilégiée, ce qui peut conduire à un grand nombre de classes, chacune comportant peu de documents. Le choix de λ n'est en pratique pas simple, et l'on a souvent recours à des valeurs qui ont été utilisées sur d'autres collections. Enfin, notons que la forme du critère ci-dessus est similaire au critère AIC, largement utilisé en sélection de modèle dans un cadre probabiliste. En fait, sous des hypothèses simples, le critère AIC (Akaike (1974a)) s'écrit, dans le cas du partitionnement avec l'algorithme des k-moyennes, sous la forme :

$$AIC(K) = SSR(K) + 2VK$$

où, rappelons-le, V désigne le nombre de termes d'indexation, c'est-à-dire la dimension des vecteurs considérés. Si ce critère est bien fondé d'un point de vue théorique, il a tendance en pratique à fournir un petit nombre de classes (car V est en général grand). Jouer avec le paramètre λ en fonction de l'application visée reste encore la meilleure solution.

Illustration Nous allons observer ici le déroulement de l'algorithme des k-moyennes sur l'exemple 1, en utilisant comme mesure de similarité le coefficient de Jaccard. Nous considérons deux classes ($K = 2$), et formulons l'hypothèse que les documents d_3 et d_4 ont été choisis comme représentants initiaux des deux classes. L'algorithme se déroule alors de la façon suivante :

Itération 1. Les documents d_1 et d_2 sont affectés à la classe la plus proche, celle de d_4 :

$$G_1^{(1)} = \{d_3\},\ G_2^{(1)} = \{d_1, d_2, d_4\}$$

Les représentants associés sont alors :

$$\begin{aligned} \mathbf{r}_1^{(1)} &= (0, 0, 1, 0, 1) \\ \mathbf{r}_2^{(1)} &= (1, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}) \end{aligned}$$

Itération 2. La similarité entre d_1 ou d_2 et $\mathbf{r}_1^{(1)}$ est nulle, alors que celle avec $\mathbf{r}_2^{(1)}$ est strictement positive. d_1 et d_2 seront donc affectés à la classe $G_2^{(2)}$. d_3 sera bien sûr affecté à la classe $G_1^{(2)}$, car sa similarité avec son représentant est maximale. Enfin, pour d_4 :

$$\text{sim}_{\text{Jaccard}}(d_4, \mathbf{r}_2^{(1)}) = \frac{5}{11} < \text{sim}_{\text{Jaccard}}(d_4, \mathbf{r}_1^{(1)}) = 0.66$$

Nous obtenons donc :

$$G_1^{(2)} = \{d_3, d_4\},\ G_2^{(2)} = \{d_1, d_2\}$$

et :

$$\begin{aligned} \mathbf{r}_1^{(2)} &= (\frac{1}{2}, 0, 1, 0, 1) \\ \mathbf{r}_2^{(2)} &= (1, \frac{1}{2}, 0, \frac{1}{2}, 0) \end{aligned}$$

Cette partition est en fait stable et n'évoluera plus aux itérations suivantes.

Implantation La mise en œuvre informatique de l'algorithme des k-moyennes ne pose pas de problème particulier si aucune optimisation n'est réalisée. Il est toutefois possible de réduire quelque peu la complexité de l'algorithme en filtrant les centroïdes considérés pour chaque document. Cette version est connue sous le nom d'*algorithme de filtrage*, et consiste à maintenir une liste des centroïdes potentiellement proches d'un ensemble de documents. Lorsque les documents sont bien séparés, cette liste est en général réduite et ne contient qu'un petit nombre de centroïdes, ce qui permet de limiter le nombre de comparaisons pour chaque document. Les algorithmes de filtrage reposent en général sur un codage des données sous forme d'arbre-kd (en anglais *kd-tree*, voir Bentley (1975)) ou d'arbre-BBD (en anglais *BBD-tree*, pour *Balanced box-decomposition tree*, voir Arya *et al.* (1998)). Une description complète d'un algorithme de filtrage efficace est donnée dans Kanungo *et al.* (2002).

Enfin, mentionnons que des implantations de l'algorithme des k-moyennes sont disponibles dans la majorité des outils de fouille de données et d'apprentissage automatique, que ces outils soient commerciaux ou libre d'accès, comme *R* [5] ou *WEKA* [6].

5. http://cran.r-project.org/ - Nous recommandons pour *R* l'utilisation du paquet *biganalytics* qui fournit une implantation plus rapide de l'algorithme des k-moyennes.
6. http://sourceforge.net/projects/weka/

6.3.2 Partitionnement hiérarchique

Le partitionnement hiérarchique (ou classification hiérarchique) vise à construire un arbre qui rend compte d'une hiérarchie possible des classes recouvrant une collection de documents. Ce partitionnement peut être réalisé de façon *bottom-up*, en créant l'arbre à partir de ses feuilles (on parle alors de méthodes agglomératives), ou *top-down*, en créant l'arbre à partir de sa racine (on parle dans ce cas de méthodes divisives). Les avantages des méthodes hiérarchiques par rapport aux méthodes à plat reposent sur le fait que les algorithmes hiérarchiques standards sont entièrement déterministes, ne nécessitent pas de fournir un nombre de classes *a priori* et produisent une structure (généralement sous forme d'arbre) des classes. En contrepartie, leur complexité est en général quadratique par rapport au nombre d'exemples (N), alors que les méthodes à plat comme l'algorithme des k-moyennes sont linéaires. Si les considérations d'efficacité prévalent, il est donc préférable d'utiliser des méthodes à plat (comme les k-moyennes). En revanche, si l'on désire obtenir un ensemble de classes structurées ou si l'on n'a aucune idée du nombre de classes, le recours aux méthodes hiérarchique devient intéressant.

Nous allons tout d'abord présenter les méthodes agglomératives qui constituent la classe de méthodes hiérarchiques la plus couramment utilisée.

Méthodes agglomératives

L'arbre créé par les méthodes agglomératives hiérarchiques (on parle en anglais de *Hierarchical Agglomerative Clustering (HAC)*) est un arbre binaire appelé *dendrogramme*. Chaque palier (identifié ici à un nœud) de l'arbre correspond à une classe et chaque étape de construction d'un nouveau palier consiste en une fusion des deux classes les plus proches. La figure 6.2 donne une illustration simple de ce processus. La collection est ici constituée de quatre documents (d_5, d_6, d_7, d_8), correspondant chacun initialement à une classe ; à chaque étape, les classes les plus proches sont fusionnées. Trois fusions (correspondant aux paliers numérotés **1**, **2** et **3** sur la figure) ont ici été réalisées :

- Les documents d_5 et d_7 sont les deux documents (ou classes) les plus proches et sont fusionnés en une nouvelle classe.
- La similarité la plus forte est ensuite obtenue entre la classe de d_8 et la classe nouvellement formée, ce qui conduit à la deuxième fusion.
- Enfin, la dernière fusion crée une classe contenant d_6 et la classe issue de la fusion précédente.

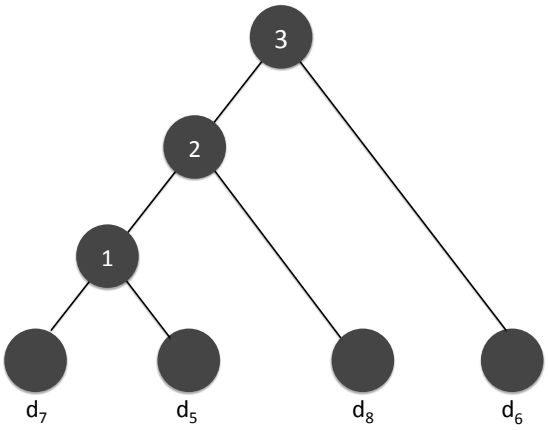

Figure 6.2 : Processus de construction d'un dendrogramme.

Le dendrogramme ainsi obtenu peut-être lu de façon très simple : la classe la plus générale contient l'ensemble de la collection ; cette classe se sépare en deux sous-classes contenant d'une part le seul document d_6 et d'autre part les autres documents ; cette dernière classe se sépare également en deux sous-classes, contenant d_8 d'une part et d_5 et d_7 d'autre part, cette dernière classe étant elle aussi divisée en deux sous-classes, l'une pour d_5 et l'autre pour d_7. s_1, s_2 et s_3 correspondent aux valeurs des similarités entre les classes fusionnées, et la hauteur d'une nouvelle classe correspond à la similarité (ou la distance) sur un axe vertical (non représenté ici par souci de simplification).

Comme on peut le remarquer, outre les similarités, que nous noterons sim, ou distances, que nous noterons $dist$, entre documents déjà abordées en début de chapitre, il est nécessaire, pour construire un dendrogramme, de définir des similarités (ou distances) entre classes pouvant contenir plusieurs documents (c'est le cas des deuxième et troisième fusions dans l'exemple ci-dessus). Ces similarités entre classes sont généralement fondées sur les similarités entre documents constituant les classes et définissent différentes méthodes d'agrégation, parmi lesquelles les trois suivantes se retrouvent dans la majorité des outils et travaux sur le partitionnement hiérarchique :

1. La méthode à lien unique (en anglais *single link*) qui considère que la similarité (respectivement distance) entre deux classes est égale à la plus grande similarité (respectivement plus petite distance) entre les docu-

ments des deux classes :

$$\begin{aligned} \text{sim}_{\text{lu}}(G_k, G_l) &= \max_{d \in G_k, d' \in G_l} sim(d, d') \\ \text{dist}_{\text{lu}}(G_k, G_l) &= \min_{d \in G_k, d' \in G_l} dist(d, d') \end{aligned}$$

Dans la méthode à lien unique, il suffit, pour que deux classes soient proches, que deux documents de chacune des classes le soient. Cette propriété du lien unique explique que cette méthode peut produire des classes qui sont relativement peu homogènes et qui ont tendance à former un dendrogramme déséquilibré, comme celui de la figure 6.2.

2. La méthode à lien complet (en anglais *complete link*) qui considère que la similarité (respectivement distance) entre deux classes est égale à la plus petite similarité (respectivement plus grande distance) entre les documents des deux classes :

$$\begin{aligned} \text{sim}_{\text{lc}}(G_k, G_l) &= \min_{d \in G_k, d' \in G_l} sim(d, d') \\ \text{dist}_{\text{lc}}(G_k, G_l) &= \max_{d \in G_k, d' \in G_l} dist(d, d') \end{aligned}$$

Dans la méthode à lien complet, il faut, pour que deux classes soient proches, que tous les documents de chacune des classes le soient ; on voit donc ici que cette méthode cherchera des classes fortement homogènes et que, pour cela, il est plus probable de construire des petites classes (à deux éléments) dans les premières étapes. On aura alors tendance à avoir un dendrogramme (trop) fortement équilibré et peu réaliste.

3. La méthode à lien moyen (en anglais *group average* ou *average link*) considère que la similarité (ou distance) entre classes est égale à la moyenne des similarités (ou distance) entre documents des deux classes :

$$\begin{aligned} \text{sim}_{\text{lm}}(G_k, G_l) &= \frac{1}{|G_k| \times |G_l|} \sum_{d \in G_k, d' \in G_l} sim(d, d') \\ \text{dist}_{\text{lm}}(G_k, G_l) &= \frac{1}{|G_k| \times |G_l|} \sum_{d \in G_k, d' \in G_l} dist(d, d') \end{aligned}$$

Dans cette méthode, deux classes sont proches si les documents les constituant sont, en moyenne, proches. Étant fondée sur une comparaison entre tous les documents des deux classes (et non pas un seul couple comme pour les méthodes précédentes), la méthode du lien moyen fournit souvent des dendrogrammes plus réalistes que ceux obtenus avec les méthodes précédentes.

Il est à noter que nous avons fourni ici la version standard du lien moyen, telle que l'on peut la trouver dans les ouvrages classiques sur la classification, comme Hastie *et al.* (2001). L'ouvrage Manning *et al.* (2008) propose pour le lien moyen une formule qui généralise celle que nous avons donnée en ce qu'elle considère la moyenne sur tous les couples de documents, y compris les documents de la même classe. En opérant ainsi, on a une meilleure garantie que les classes obtenues soient homogènes. Enfin, nous n'avons pas présenté ici la méthode de Ward car son fonctionnement général (algorithme, complexité, monotonicité...) est le même que celui des méthodes ci-dessus. De plus, elle est très similaire à la méthode à lien moyen fondée sur un carré de la distance entre documents. Nous renvoyons les lecteurs intéressés par cette méthode à Saporta (2011).

La formule de Lance-Williams Les formules d'agrégation présentées ci-dessus sont en fait des cas particuliers d'une formule plus générale due à Lance et Williams (Lance et Williams (1967)) et qui prend la forme suivante, où la classe G_i est considérée comme résultant de la fusion des classes G_i^1 et G_i^2 (nous donnons ici la formulation avec une similarité, la même formulation existant pour une distance) :

$$\begin{aligned} \text{sim}(G_k, G_l) &= \alpha_1 \text{sim}(G_k^1, G_l) + \alpha_2 \text{sim}(G_k^2, G_l) + \beta \text{sim}(G_k^1, G_l^2) \\ &\quad + \gamma |\text{sim}(G_k^1, G_l) - \text{sim}(G_k^2, G_l)| \end{aligned} \tag{6.6}$$

Cette formule est bien générale car toute fusion de classes (à part celles n'impliquant que des classes réduites à un seul document, mais ce cas est traité de façon triviale) met en jeu au moins une classe résultant de la fusion de deux classes. Le tableau 6.1 résume comment les formules d'agrégation à lien simple, complet et moyen dérivent de l'équation 6.6.

Mesure d'agrégation	α_1	α_2	β	γ
lien simple	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
lien complet	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
lien moyen	$\frac{G_k^1}{\lvert G_k^1 \rvert + \lvert G_k^2 \rvert}$	$\frac{G_k^1}{\lvert G_k^1 \rvert + \lvert G_k^2 \rvert}$	0	0

Tableau 6.1 : Paramètres de la formule de Lance-Williams pour différentes mesures d'agrégation.

L'avantage des mesures d'agrégation pouvant s'écrire sous la forme de Lance-Williams est que le calcul des similarités (ou distances) entre une classe nouvellement créée et les autres peut être réalisé en temps constant puisque ce calcul repose sur des similarités (ou distances) déjà connues.

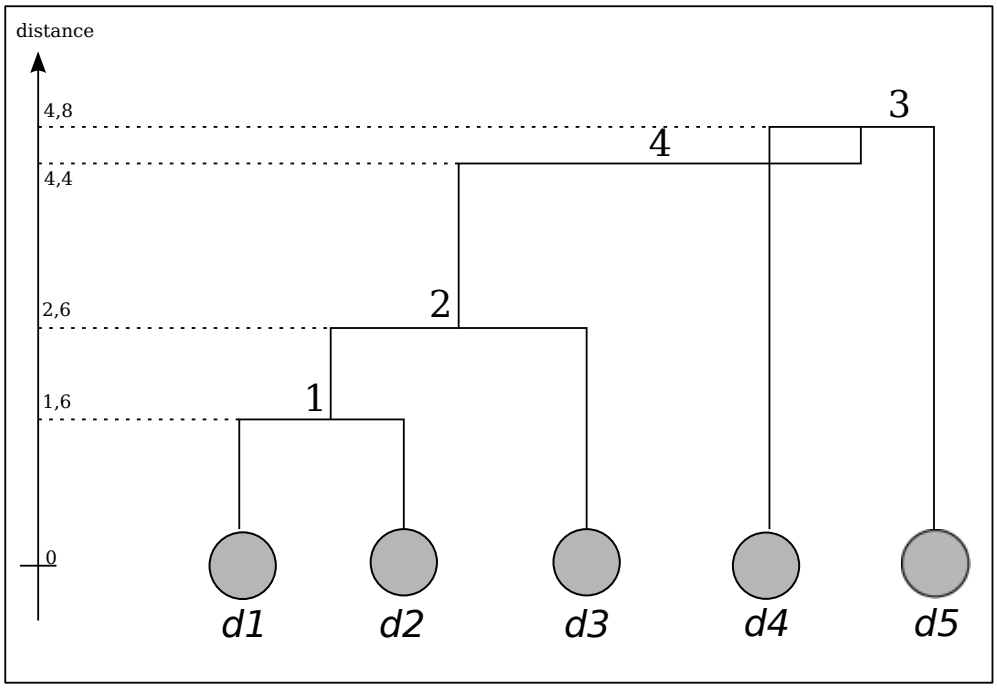

Figure 6.3 : Exemple de dendrogramme obtenu avec une méthode non monotone.

Monotonicité des méthodes d'agrégation Il est possible de définir une notion de monotonicité pour les méthodes d'agrégation, liée au fait qu'à chaque étape de fusion la similarité (respectivement la distance) diminue (respectivement augmente). La figure 6.3 illustre cette notion avec un dendrogramme obtenu par une méthode non monotone. Si les trois premières fusions sont bien telles que la distance entre les classes fusionnées augmente à chaque étape, la distance mise en jeu dans la quatrième fusion est inférieure à celle considérée dans la troisième fusion (bien sûr, la distance entre d_4 ou d_5 et la classe contenant d_1, d_2 et d_3 à l'étape 3 est supérieure à la distance entre d_4 et d_5, sinon la troisième fusion n'aurait pas lieu).
Cette notion de monotonicité peut se formaliser de la façon suivante.

Définition 3. Monotonicité. *Soit $G^{(r)} = G_1^{(r)} \cup G_2^{(r)}$ la classe obtenue à l'étape r de construction d'un dendrogramme par une méthode d'agrégation $\mathcal{M}$ et soit $G^{(r+1)} = G_1^{(r+1)} \cup G_2^{(r+1)}$ la classe obtenue à l'étape $r+1$. Nous dirons que $\mathcal{M}$ est monotone si et seulement si :*

$$sim(G_1^{(r)}, G_2^{(r)}) \geq sim(G_1^{(r+1)}, G_2^{(r+1)})$$

ou de façon équivalente, si et seulement si :

$$dist(G_1^{(r)}, G_2^{(r)}) \leq dist(G_1^{(r+1)}, G_2^{(r+1)})$$

à chaque étape r.

Il est facile de voir que le nombre d'étapes requises pour construire un dendrogramme à partir de N documents est au plus $N-1$ (il est même en général égal à $N-1$). En effet, lorsque la construction du dendrogramme est complète, c'est-à-dire lorsqu'il est possible de regrouper tous les documents dans une même classe [7], le problème se ramène, après la première étape (fusion de deux documents en une seule classe), à la construction d'un dendrogramme à partir de $N-1$ objets. Le nombre d'étapes pour N objets est donc égal au nombre d'étapes pour $N-1$ objets plus 1. On obtient ainsi par récurrence la relation souhaitée.

La notion de monotonicité ci-dessus est importante car elle garantit une certaine optimalité des classes obtenues : chaque étape est optimale non seulement parce qu'elle opère la meilleure fusion possible à un instant donné (par définition de la construction du dendrogramme) mais aussi car il n'y aura pas de fusion dans les étapes suivantes qui donnerait une meilleure distance ou similarité. Les trois méthodes d'agrégation que nous avons considérées (lien simple, lien complet et lien moyen) sont monotones au sens de la définition précédente (voir exercice 6.5).

Algorithme de partitionnement hiérarchique agglomératif Nous donnons ci-dessous un algorithme général de partitionnement hiérarchique compatible avec les trois méthodes d'agrégation retenues ici (mais bien d'autres méthodes peuvent également être prises en compte). Cet algorithme, qui opère en deux passes (construction de la matrice de similarité entre documents et des structures de données utiles, puis construction proprement dite du dendrogramme), repose sur l'utilisation de files de priorité qui, d'une part, contiennent, pour chaque classe, l'ensemble des similarités, triées par ordre décroissant, avec les autres classes et, d'autre part, permettent de supprimer, d'insérer ou de trier des éléments de façon efficace (en $O(\log(n))$, où n représente le nombre d'éléments dans la file). Ces files de priorité sont définies pour chaque document (et donc pour chaque classe initiale) de la collection, puis mises à jour au fur et à mesure de la construction du dendrogramme. Elles sont notées $P[i]$ où i est un indice de document. $P[i].MAX$.sim désigne la similarité maximum entre la classe i et les autres classes et $P[i].MAX$.index représente l'indice de

7. Ceci n'est pas toujours acquis car il est possible, mais rare en pratique, qu'un document ait une similarité nulle avec tous les autres documents de la collection par exemple.

la classe la plus proche de i. L'ensemble des classes actives est conservé dans un tableau ($I[i]$) et les informations relatives à i_1 sont mises à jour lorsque les classes i_1 et i_2 sont fusionnées (la classe i_2 devient alors inactive). Nous supposons de plus (par souci de simplification) que la mesure de similarité entre documents utilisée est symétrique, et nous noterons $\mathcal{M}$ la méthode d'agrégation considérée (lien unique...). Enfin, les cas d'égalité entre mesures sont supposés résolus de façon déterministe, par exemple en choisissant systématiquement la classe d'indice le plus petit pour fusion.

ALGORITHME 18. Algorithme de partitionnement hiérarchique agglomératif

Entrée : $\mathcal{C} = \{d_1, \cdots, d_N\}$, collection de documents
pour chaque $i \in \{1, \ldots, N\}$ **faire**
  pour chaque $j \in \{1, \ldots, N\}$ **faire**
    $C[i][j] \leftarrow sim(d_i, d_j)$; //Initialisation de la matrice de similarité
  fin
  $I[i] \leftarrow 1$; //indicateur de classe active
  construire $P[i]$, file de priorité pour d_i triée suivant $C[i][]$;
fin
$L \leftarrow \emptyset$;
//Construction du dendrogramme
pour chaque $k \in \{1, \ldots, N-1\}$ **faire**
  $i_1 \leftarrow \text{argmax}_{i:I[i]=1} P[i].MAX$.sim;
  $i_2 \leftarrow P[i_1].MAX$.index;
  $L \leftarrow L.(i_1, i_2)$; //ajout de (i_1, i_2) à l'ensemble des fusions
  $I[i_2] \leftarrow 0$;
  supprimer $C[i_1][i_2]$ de $P[i_1]$;
  pour chaque i *tel que* $I[i] = 1 \wedge i \neq i_1$ **faire**
    supprimer $C[i][i_1]$ de $P[i]$ et $P[i_1]$ et $C[i][i_2]$ de $P[i]$;
    $C[i][i_1], C[i_1][i] \leftarrow \text{sim}_{\mathcal{M}}(i, (i_1, i_2))$;
    insérer (avec tri) $C[i][i_1]$ dans $P[i]$ et $P[i_1]$;
  fin
fin
Sortie : la liste des fusions L.

L'algorithme 18 retourne la liste des fusions qui ont été effectuées. Il est bien sûr possible de retourner également les valeurs des similarités entre classes fusionnées de façon à pouvoir construire le dendrogramme complet (c'est-à-dire avec l'axe des similarités/distances). La complexité de l'étape d'initialisation est dominée par la double boucle sur N éléments, et vaut donc $O(N^2)$. La deuxième étape comprend une boucle sur $N-1$ éléments dans

laquelle nous commençons par récupérer la classe active ayant la plus grande similarité avec une autre classe (complexité en $O(N)$), avant d'effectuer un certain nombre d'opérations à temps constant puis de répéter, au plus $N-2$ fois, des opérations de suppression et d'insertion avec tri dans une file de priorité contenant au plus $N-1$ éléments. Ces dernières opérations ont une complexité de l'ordre de $O(\log N)$ et sont répétées $(N-1)\times(N-2)$ fois. La complexité globale de cet algorithme est donc : $O(N^2 \log N)$. Enfin, il existe un algorithme plus performant pour la méthode d'agrégation à lien unique (en $O(N^2)$), fondé sur l'utilisation de tableaux NBM (voir exercice 6.6).

Nous allons illustrer l'algorithme 18 sur l'exemple donné en début de chapitre, en prenant pour similarité entre documents le coefficient de Dice (page 160), et comme méthode d'agrégation la méthode à lien unique. Après l'étape d'initialisation, nous avons les structures de données suivantes :

$$C^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 0,25 & 0 & 0,20 \\ 0,25 & 1 & 0 & 0,25 \\ 0 & 0 & 1 & 0,40 \\ 0,20 & 0,25 & 0,40 & 1 \end{pmatrix} \end{array}$$

$$I^{(0)} = \begin{array}{|cccc|} \hline 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 1 \\ \hline \end{array}$$

$$P^{(0)}[1] = \begin{array}{|ccc|} \hline 2 & 4 & 3 \\ \hline 0,25 & 0,20 & 0 \\ \hline \end{array}$$

$$P^{(0)}[2] = \begin{array}{|ccc|} \hline 1 & 4 & 3 \\ \hline 0,25 & 0,25 & 0 \\ \hline \end{array}$$

$$P^{(0)}[3] = \begin{array}{|ccc|} \hline 4 & 1 & 2 \\ \hline 0,40 & 0 & 0 \\ \hline \end{array}$$

$$P^{(0)}[4] = \begin{array}{|ccc|} \hline 3 & 2 & 1 \\ \hline 0,40 & 0,25 & 0,20 \\ \hline \end{array}$$

Les classes sélectionnées pour fusion lors de la première exécution sont les classes 3 et 4 ($i_1 = 3$, $i_2 = 4$). La mise à jour des structures de données conduit à :

$$I^{(1)} = \begin{array}{|cccc|} \hline 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 0 \\ \hline \end{array}$$

$$C^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 1 & 0,25 & 0,20 \\ 0,25 & 1 & 0,25 \\ 0,20 & 0,25 & 1 \end{pmatrix} \end{array}$$

$$P^{(1)}[1] = \begin{array}{|cc|}\hline 2 & 3 \\ \hline 0,25 & 0,20 \\ \hline \end{array}$$

$$P^{(1)}[2] = \begin{array}{|cc|}\hline 1 & 3 \\ \hline 0,25 & 0,25 \\ \hline \end{array}$$

$$P^{(1)}[3] = \begin{array}{|c|}\hline 1 \\ \hline 0,25 \\ \hline \end{array}$$

$$L = \{(3,4)\}$$

Puis, à la deuxième fusion, ce sont les classes 1 et 2 qui sont sélectionnées ($i_1 = 1$, $i_2 = 2$), conduisant à la mise à jour :

$$I^{(2)} = \begin{array}{|cccc|}\hline 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 \\ \hline \end{array}$$

$$C^{(2)} = \begin{array}{c} \\ 1 \\ 3 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 3 \end{array} \\ \begin{pmatrix} 1 & 0,25 \\ 0,25 & 1 \end{pmatrix} \end{array}$$

$$P^{(2)}[1] = \begin{array}{|c|}\hline 3 \\ \hline 0,25 \\ \hline \end{array}$$

$$P^{(2)}[3] = \begin{array}{|c|}\hline 1 \\ \hline 0,25 \\ \hline \end{array}$$

$$L = \{(3,4),(1,2)\}$$

Enfin, la dernière étape fusionnera les classes 1 et 3, conduisant à : $L = \{(3,4),(1,2),(1,3)\}$.

Sélection du nombre de classes Comme nous l'avons déjà signalé, un des avantages du partitionnement hiérarchique agglomératif est que l'utilisateur n'a pas à spécifier un nombre de classes *a priori*. Cependant, dans certaines applications, il peut être plus intéressant de fournir une vue partielle du dendrogramme, resposant sur un nombre restreint de classes. Il y a quatre grands types d'approches pour obtenir cette vue partielle :

- Couper le dendrogramme à un niveau donné de similarité. On considère alors que tous les regroupements effectués après ce niveau de similarité ne sont pas intéressants car rapprochant des classes non suffisamment proches les unes des autres. Cette approche demande bien sûr à l'utilisateur de fournir un seuil sur la similarité, ce qui n'est pas toujours facile.

– Couper le dendrogramme au niveau où la différence entre les valeurs de similarité de deux fusions successives est maximale. On estime alors que, avant cette coupe, les similarités des classes fusionnées étaient relativement proches les unes des autres ; au niveau de la coupe, une décroissance plus marquée de la similarité est observée, ce qui suggère une moins forte cohésion des classes. Aucun paramètre n'est ici nécessaire.

– Utiliser le critère défini par l'équation 6.5 comme dans le cas du partitionnement à plat ; les mêmes remarques que celles déjà formulées dans le cas du partitionnement à plat s'appliquent bien sûr ici.

– Couper le dendrogramme de façon à se reposer sur K classes, K étant un nombre de classes spécifié par l'utilisateur ; on se retrouve ici dans un cas similaire à celui du partitionnement à plat avec les k-moyennes car on demande à l'utilisateur de choisir un nombre de classes à l'avance. On peut ensuite partir de la racine et trouver la coupe horizontale qui fournit le nombre de classes souhaitées.

Méthodes divisives

Les méthodes divisives (ou méthodes *top-down*) reposent sur l'utilisation récursive d'une méthode de partitionnement à plat (processus que nous avons déjà évoqué pour l'extension hiérarchique des k-moyennes page 168). Un désavantage de cette approche est bien sûr qu'elle nécessite en général de fournir le nombre de classes à chaque sous-division. En revanche, ce genre d'approche s'avère intéressant dès lors que l'on ne souhaite qu'un nombre fini, peu élevé, de niveaux de sous-divisions, depuis une classe générale contenant toute la collection jusqu'à quelques classes plus spécifiques. Le niveau de granularité souhaité est ici entièrement contrôlé, contrairement à ce qui se passe pour les approches agglomératives.

Un autre avantage de ces méthodes est que l'arbre obtenu n'est pas nécessairement binaire, ce qui permet de rendre mieux compte des possibles structurations d'une collection de documents. Enfin, en prenant en compte l'ensemble des documents pour effectuer un partitionnement, on se repose sur une information globale, ce que ne font pas les méthodes agglomératives qui consistent en des fusions certes optimales mais locales. Dans de nombreux cas, utiliser une méthode divisive, fondée par exemple sur l'algorithme des k-moyennes, peut s'avérer plus intéressant que l'utilisation de méthodes agglomératives fournissant un dendrogramme.

6.4 Évaluation

Des quantités comme la somme des carrés des résidus (voir équation 6.2) peuvent bien sûr être utilisées pour mesurer la qualité d'un partitionnement. Toutefois, il n'est pas garanti qu'un partitionnement qui obtient un bon score sur des quantités de ce type soit le plus utile dans le cadre d'une application donnée, et il est plus judicieux d'évaluer l'apport du partitionnement au sein d'une application donnée. Par exemple, dans le cadre de la recherche d'information, on peut vouloir utiliser un partitionnement pour aider l'utilisateur à mieux comprendre une liste de résultats et à améliorer sa recherche (voir section 6.5). Dans ce cadre, la qualité d'un partitionnement peut être mesurée par le temps moyen gagné lors de différentes sessions de recherche.

Une des principales utilisations du partitionnement est la constitution d'un ensemble de classes, qui pourront servir par la suite à des fins de catégorisation. On peut alors évaluer un algorithme de partitionnement en évaluant les résultats qu'il fournit sur un jeu de données déjà catégorisées. Cette pratique est très courante ; elle repose sur l'hypothèse, souvent non formulée, que la représentation des documents retenues est en adéquation avec le type de catégories prédéfinies (les attributs utilisés par exemple pour une catégorisation thématique sont différents de ceux retenus pour une catégorisation en genre - genre policier, journalistique...). Deux mesures sont en général employées pour comparer les résultats d'un partitionnement avec des catégories prédéfinies (telles que celles de la collection Reuters présentée au chapitre précédent) : *l'indice de pureté* et *l'information mutuelle normalisée*.

L'indice de pureté vise à quantifier le fait que les classes d'une partition ne regroupent que des documents de la même catégorie. Soit G une partition d'une collection et C un ensemble de catégories définies sur cette partition (G est obtenue à l'aide d'un algorithme de partitionnement, sans tenir compte des catégories de C). L'indice de pureté (en anglais *purity*) est défini par :

$$\text{pure}(G, C) = \frac{1}{N} \sum_{k} \max_{l} |G_k \cap C_l| \tag{6.7}$$

Pour chaque classe G_k de la partition, on recherche donc la catégorie C_l de C qui contient le plus de documents de G_k, et on calcule un score normalisé de la taille de l'intersection entre G_k et C_l. Si tous les documents de G_k sont contenus dans C_l, alors G_k est pure au sens où elle ne contient que des documents de la même catégorie. Si toutes les classes de la partition sont pures, le score de pureté est maximal et vaut 1. Cette valeur maximale est nécessairement atteinte lorsqu'il y a autant de classes dans la partition que de

documents. On voit donc que le score de pureté ne peut pas être utilisé pour sélectionner le nombre de classes, et que son usage doit être restreint à la comparaison de partitions (ou d'algorithmes de partitionnement) comportant le même nombre de classes (dans la majorité des études, le nombre de classes considérées est égal au nombre de catégories de C).

L'information mutuelle normalisée est définie par :

$$\text{IMN}(G, C) = \frac{2 \times I(G, C)}{H(G) + H(C)} \tag{6.8}$$

où I désigne l'information mutuelle (voir chapitre 5) et H l'entropie. Ces deux quantités prennent ici la forme suivante :

$$\begin{aligned} I(G, C) &= \sum_k \sum_l P(G_k \cap C_l) \log \frac{P(G_k \cap C_l)}{P(G_k)P(C_l)} \\ &= \sum_k \sum_l \frac{|G_k \cap C_l|}{N} \log \frac{N|G_k \cap C_l|}{|G_k||C_l|} \end{aligned}$$

et :

$$\begin{aligned} H(G) &= -\sum_k P(G_k) \log P(G_l) \\ &= -\sum_k \frac{|G_k|}{N} \log \frac{|G_k|}{N} \end{aligned} \tag{6.9}$$

L'intérêt de cette formule est qu'elle atteint son maximum (qui vaut 1) lorsque les classes de la partition et les catégories sont identiques. L'effet du nombre de classes sur l'indice de pureté est ici gommé par la normalisation par l'entropie.

6.5 Applications à l'accès à l'information

Les applications du partitionnement à l'accès à l'information touchent à peu près tous les aspects de ce domaine, comme l'identification des clients qui ont des profils d'achat similaires, l'identification de thèmes dans une collection de documents ou le groupement de machines suivant leurs fichiers logs pour maintenance/dépannage. Le partitionnement joue de plus un rôle majeur dans l'aide à la navigation et la visualisation.

Sur ce dernier point, plusieurs outils ont été développés pour permettre une meilleure visualisation des résultats d'une recherche [8], l'idée étant de fournir aux utilisateurs une cartographie des documents sur un thème donné, l'utilisateur pouvant alors affiner sa requête suivant les thématiques présentes dans la collection sur laquelle porte la recherche. Une vision intéressante de la navigation dans des collections de documents, couplée à une problématique de recherche d'information, est celle développée par l'approche éparpiller/grouper (en anglais *scatter/gather*) (voir Cutting *et al.* (1992)). Dans cette approche, le processus de recherche d'information est complété par un processsus de navigation (en anglais *browsing*) qui repose sur un algorithme de partitionnement augmenté de procédures d'éclatements et de regroupements de classes qui permettent une interaction plus forte avec l'utilisateur. Dans l'étude originale, une première procédure de partitionnement hiérarchique agglomératif, fondée sur le lien moyen, est utilisée pour obtenir des centres initiaux de classes. Une procédure de type k-moyennes est ensuite utilisée pour affiner ce premier jeu de classes.

De manière plus générale, le partitionnement est largement utilisé dès lors que l'on veut obtenir une première vue et une première analyse d'un jeu de données. Les méthodes que nous avons présentées ici sont très générales et peuvent être appliquées sur des types de données très différents.

8. Voir par exemple http://search.yippy.com/