AHClassif
Package R pour la classification ascendante hiérarchique

Tristan Margate, Malik Sari

Abstract

L'un des principaux outils en analyse statistique et en data mining est le clustering, permettant de trouver des similarités entre des données et d'être en mesure de synthétiser l'information qu'elles contiennent. L'un des principaux problèmes de ces algorithmes de clustering est qu'ils peuvent s'avérer extrêmement coûteux en terme de temps, et l'augmentation sans cesse des volumes de ces données nous impose donc de construire des algorithmes dont le coût temporel devient de plus en plus faible. Ce rapport présente différents algorithmes de classification ascendant hiérarchiques et compare leur complexité temporelle afin de pouvoir déterminer lequel s'avère être le moins coûteux temporellement, dans l'état de l'art actuel, pour réaliser un clustering.

Contents

1 Introduction

Lorsque nous avons en notre possession des données et que nous voulons effectuer une description de celles-ci, de multiples méthodes se présentent à nous afin d'exercer une analyse statistique pertinente. Dans le cadre d'un problème d'apprentissage non supervisé, il est courant d'utiliser des techniques visant à regrouper les données dites "similaires" entre elles, créant ainsi des sous-groupes de données partitionnant notre jeu de données initiale, ces techniques sont appelés techniques de **clustering** et sont extrêmement utilisées en Data Mining. Ces sous-groupes, que nous appellerons dans la suite **clusters**, sont utilisés pour synthétiser l'information qui se trouvent dans les données initiales. Deux principales approches existent pour effectuer un clustering :

- Le clustering par partitionnement
- Le clustering hiérarchique

Les algorithmes fonctionnant par clustering hiérarchique sont des algorithmes itératives, fonctionnant de manière récurrente, cherchant des clusters en utilisant les clusters précédemment établis, tandis que les algorithmes de clustering par partitionnement essaient de trouver tous les clusters à la fois, cela se traduit par le fait de définir dès le début un nombre de clusters K qui affectera beaucoup le résultat, vous pourrez trouver plus d'explications dans l'article [2] concernant ce type de clustering, dans tout ce rapport, nous étudierons uniquement les algorithmes à clustering hiérarchique.

1.1 Clustering hiérarchique

Comme indiqué précédemment, les algorithmes de clustering hiérarchique sont des algorithmes itératives partitionnant nos données en clusters à chaque itération. Cependant, là aussi, un clustering hiérarchique peut s'effectuer de deux manières différentes :

- Un clustering hiérarchique ascendant
- Un clustering hiérarchique descendant

En effet, il est possible de considérer dès le début que toutes nos données font parti d'un seul et même cluster, et nous allons le diviser itérativement en clusters afin d'arriver à nos fins. Cette méthode est le clustering hiérarchique descendant, plus communément appelé dans la littérature clustering hiérarchique divisive (**divisive hierarchical clustering** ou encore **clustering top-down**). Là encore, nous ne nous attarderons pas sur ce type d'approche dans ce rapport, cet article [4] détaille cependant ce type d'algorithme. Nous nous pencherons ici sur le clustering hiérarchique ascendant (**agglomerative hierarchical clustering** ou encore **clustering bottom-up**), aussi nommé **classification ascendante hiérarchique**, qui consiste à effectuer en quelque sorte l'opposé du clustering top-down, c'est-à-dire de considérer chacune des données comme des clusters différents dès le début de l'algorithme, puis à chaque itération de les regrouper en fonction de leur similarité afin de créer nos clusters représentatif de nos données. L'un des principaux problèmes majeurs de ces types d'algorithmes est qu'ils peuvent s'avérer extrêmement coûteux en termes de complexité temporelle, et en raison de l'augmentation du nombre de données dans les dernières décennies, il est nécessaire de trouver des algorithmes le moins coûteux possible pour réaliser ce type de classification. Dans la suite, nous exposerons plusieurs algorithmes de classification ascendant hiérarchique afin de comparer leur performances sur des données et ainsi en déduire quel algorithme est le plus adéquat dans une situation donnée.

Maintenant que les termes utilisés sont définis et vraisemblablement claires, il est temps de définir les notations ainsi que le contexte que nous allons utiliser dans toute la suite de ce rapport pour expliquer les algorithmes présents dans ce package **R**.

1.2 Contexte et notations

Nous noterons $\mathcal{X}$ l'ensemble des données que nous possédons. Soit $n \in \mathbb{N}^*$ et $d \in \mathbb{N}^*$, nos données seront représentés par n vecteurs de l'espace vectoriel euclidien $\mathbb{R}^d$, que nous noterons $(x_i)_{[\![1,n]\!]}$, nous appellerons x_i l'individu numéro i, et nous noterons $x_{i,k}$ la k-ème composante du vecteur x_i $\forall k \in [\![1, d]\!]$.

Un cluster sera défini simplement comme étant la réunion des individus présents dans celui-ci, nous les noterons $(C_j)_{j\in[\![1,K]\!]}$ avec K le nombres de clusters présent à un instant t dans notre partition, le cardinal d'un cluster C_j sera noté $|C_j|$, on a donc naturellement

- $\forall (j,k) \in [\![1,K]\!]^2,\ \ j \neq k \implies C_j \cap C_k = \emptyset$
- $\mathcal{X} = \bigcup_{j \in [\![1,K]\!]} C_j$

La distance euclidienne sera la distance usuelle lorsqu'il s'agira d'étudier la distance entre deux individus, c'est-à-dire que sauf mention contraire, on définit la distance D suivante :

$$\begin{array}{rccc} D & : & (\mathbb{R}^d)^2 & \to & R^+ \\ & & (x_i, x_j) & \mapsto & \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2} \end{array}$$

Ainsi, dès la première itération de nos futurs algorithmes, en se basant sur la description de la classification ascendante hiérarchique, on peut définir une similarité entre nos individus, permettant de mesurer à quel points deux individus sont proches, juste en se basant sur cette distance D définie. En effet, nous pouvons à présent définir une matrice des distances, appelé **matrice de dissimilarité**, noté $\mathcal{D}$, qui contiendra les distances entre les individus de la façon suivante :

$$\forall (i,j) \in [\![1,n]\!],\ \ \mathcal{D}_{i,j} = D(x_i, x_j)$$

$\mathcal{D}$ est donc une matrice de taille $n \times n$ symétrique, contenant des 0 sur la diagonale.

$$\mathcal{D} = \begin{pmatrix} 0 & D(x_1,x_2) & \dots & \dots & \dots & \dots & D(x_1,x_n) \\ D(x_2,x_1) & 0 & \ddots & & & \dots & D(x_2,x_n) \\ \vdots & \ddots & \ddots & & & \vdots & \\ \vdots & & & & & & \vdots \\ \vdots & & & & & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & D(x_{n-1},x_n) \\ D(x_n,x_1) & D(x_n,x_2) & \dots & \dots & \dots & D(x_n,x_{n-1}) & 0 \end{pmatrix}$$

Afin de comparer les performances de nos algorithmes avec les algorithmes classiques étudiés dans l'état de l'art, nous effectuerons nos clustering dans un cadre d'apprentissage supervisé, c'est-à-dire que nos données seront d'avance labélisés afin de remarquer si nos clustering sont bien effectués. Nous prendrons comme exemple principal le célèbre jeu de données **Iris** de Fisher, contenant 50 fleurs de chacune des trois catégories différentes :

- les setosas
- les versicolors
- et les virginicas

Dans ce jeu données se trouvent quatre variables explicatives:

1. La longueur du pétale
2. La largeur du pétale
3. La longueur du sépale
4. La largeur du sépale

L'idée des différents algorithmes que nous allons introduire sera donc de pouvoir identifier ces trois catégories de fleurs en trois clusters en se basant sur l'information fournis par ces quatre variables explicatives.

Au cours de la formation des clusters, il est possible d'observer un arbre d'arborescence, aussi appelé dendogramme, qui est un schéma représentant nos clusters à un instant t, comme nous le montre la figure (1) nous venant de [3] avec ici 6 indidivus (et donc 6 clusters) initialement, puis on voit les clusters se réunir et se former itérativement.

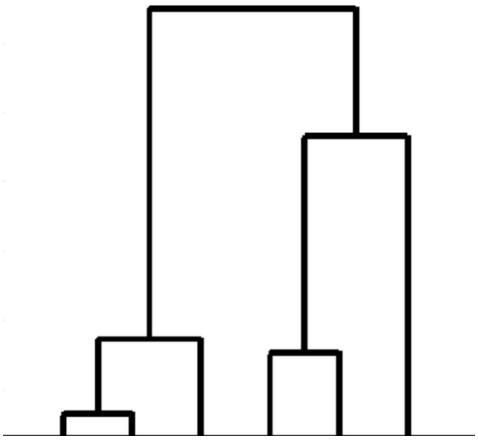


Figure 1: Dendogramme

Cependant, une gêne s'installe, car nous avons défini plus haut des distances entre des individus, mais comme défini dans l'introduction, il faudra par la suite définir des distances entre des clusters afin de savoir quels clusters sont similaires, dans le but de les fusionner à leur tour pour en créer un nouveau cluster. Il nous faut donc définir une distance entre clusters

1.3 Formule de Lance-Williams

La nécessité de construire une distance entre les clusters nous ammène à considérer plusieurs possibilités de distances, en effet, il est possible de définir une distance entre deux clusters C_i et C_j de différentes manières :

- $D(C_i, C_j) = \min\{D(x, y) : x \in C_i, y \in C_j\}$ (single linkage), les voisins les plus proches
- $D(C_i, C_j) = \max\{D(x, y) : x \in C_i, y \in C_j\}$ (complete linkage), les voisins les plus éloignés
- $D(C_i, C_j) = \frac{\sum_{x \in C_i, y \in C_j} D(x,y)}{|C_i| \times |C_j|}$ (average linkage), distance moyenne entre les individus.

Il en existe encore d'autres que l'on peut retrouver dans [7].

Il n'y a pas de définition objective pour définir une telle distance, c'est pourquoi, en 1967, Lance et Williams ont défini une formule itérative permettant de regrouper plusieurs de ces définitions de distances entre les clusters sous une seule formule : **La formule de Lance Williams**, elle se définit de la façon suivante :

Soit C_i et C_j deux clusters, ainsi que $C_{ij} = C_i \cup C_j$ la réunion de ces deux-ci, et C_k un autre cluster. Soit $(\alpha_i, \alpha_j, \beta, \gamma) \in \mathbb{R}^4$, on définit une distance de Lance-Williams, noté D_{LW} entre le nouveau cluster C_{ij} et C_k de la manière suivante :

$$D_{LW}(C_{ij}, C_k) = \alpha_i D_{LW}(C_i, C_k) + \alpha_j D_{LW}(C_j, C_k) + \beta D_{LW}(C_i, C_j) + \gamma |D_{LW}(C_i, C_k) - D_{LW}(C_j, C_k)|$$

Avec pour convention, que si la distance de Lance-Willams est calculée entre deux singletons, alors il s'agit de la distance euclidienne, c'est-à-dire que si $C_i = \{x_i\}$ et $C_j = \{x_j\}$, alors $D_{LW}(C_i, C_j) = D(x_i, x_j)$. Cette formule nous permet de retrouver les différents types de distances énoncés précédemment ainsi que bien d'autres juste en évaluant les valeurs des paramètres réels $\alpha_i, \alpha_j, \beta$ et γ, on peut voir cela à travers le tableau 1.1, plus détaillé dans [2].

Maintenant que nos distances entre clusters sont désormais définis, nous pouvons passer aux algorithmes afin de réaliser nos classifications ascendantes hiérarchiques.

2 Algorithme classique, dit "naïf"

Dans cette partie, nous allons introduire notre premier algorithme de classification ascendante hiérarchique, très connu et largement utilisé pour effectuer du clustering en apprentissage non supervisé.

Méthodes	α_i	α_j	β	γ
single	$1/2$	$1/2$	0	-1/2
complete	$1/2$	$1/2$	0	$1/2$
average	$\frac{\lvert C_i\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$	$\frac{\lvert C_i\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$	0	0
centroid	$\frac{\lvert C_i\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$	$\frac{\lvert C_i\rvert}{\lvert C_i\rvert+\lvert C_j\rvert}$	$-\frac{\lvert C_i\rvert\times\lvert C_j\rvert}{(\lvert C_i\rvert+\lvert C_j\rvert)^2}$	0
median	$1/2$	$1/2$	-1/4	0

Table 1.1: Formule de Lance-Williams : méthodes et paramètres

Nous allons dans un premier temps définir son fonctionnement de façon générale, puis nous coderons cet algorithme et nous exposerons sa complexité temporelle.

2.1 Heuristique

Une manière assez naturelle pour réaliser une classification ascendante hierarchique serait de calculer toutes les distances entre les individus puis de regrouper les deux individus pour lesquels la distance est minimale, de ce fait nous créons un cluster, puis nous réalisons la démarche précédente, en recalculant les distances entre le cluster généré et les individus, puis on assemble etc... Il est important de noter que nous réalisons dans ce cas là une seule fusion de cluster à la fois, car en effet, fusionner plusieurs clusters dans une même itération peut nous faire manquer une fusion de clustering plus intéressante, si on considère C_i, C_j, C_k et C_l quatre clusters, alors on peut avoir les relations suivantes :

- $\min_{a,b} D_{LW}(C_a, C_b) = D_{LW}(C_i, C_j)$
- $D_{LW}(C_k, C_l) < D_{LW}(C_k, C_i)$
- $D_{LW}(C_k, C_l) < D_{LW}(C_k, C_j)$
- mais $D_{LW}(C_k, C_i \cup C_j) < D_{LW}(C_k, C_l)$

Ainsi, il est judicieux de fusionner en premier lieu les clusters C_i et C_j, puis d'attendre la prochaine itération, avec le calcul de la nouvelle matrice de dissimilarité $\mathcal{D}$, afin de fusionner le cluster C_k (on rappelle que même les individus sont considérés comme des clusters, ainsi tout peut se calculer avec la formule de Lance-Williams directement). En créant un algorithme comme ceci, on peut alors définir un critère d'arrêt comme étant un nombre de clusters à atteindre, si l'on est dans un cas supervisé, ou bien dans un cas non supervisé si un a priori sur les données permets de le faire ou alors différentes méthodes de critères d'arrêt classiques utilisés dans l'algorithme des K-means, comme la méthode du coude par exemple décrite dans [6], pouvant mettre fin à l'algorithme.

Nous nous concentrons ici avec un nombre fixe de clusters à atteindre comme critère d'arrêt, on note p le nombre de clusters désiré à la fin de notre algorithme. Pour pouvoir stocker les clusters, nous emploierons la méthode suivante : nous allons définir une matrice P, que l'on nommera matrice des clusters, et qui contiendra pour chaque itération le numéro du cluster dont appartiendra un individu donnée, cette matrice P sera donc de taille $p \times n$, chaque ligne sera associée à une itération de l'algorithme, et chaque colonne à un individu. Afin d'illuster notre propos, prenons l'exemple de notre dendogramme à la figure 1, on aura la matrice P suivante :

$$P = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 1 & 2 & 3 & 4 & 5 \\ 1 & 1 & 2 & 3 & 3 & 4 \\ 1 & 1 & 1 & 2 & 2 & 3 \\ 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

On peut placer trois remarques :

1. Il ne faut pas oublier de descendre les numéros des clusters à chaque fusion de clusters, car on en perd un à chaque itération

2. Ici on a $p = n$, notre algorithme sera codé de façon à ce que ce soit le cas, cependant une simple condition d'arrêt peut être formulée pour $p < n$

3. Les valeurs de la diagonale de $\mathcal{D}$ étant nulles, il est judicieux de les mettre égale à l'infini pour éviter que la recherche de minimum pointe vers le cas triviale : pour un individu donné, l'individu le plus proche est évidemment lui-même

Maintenant que notre algorithme est définie, nous pouvons en formuler un "pseudo-code" assez simple afin de mettre en évidence les différentes étapes successives de celui-ci :

Algorithm 1 Algorithme classique

procedure
    On choisit un type de linkage
    On initialise la matrice des clusters P
    Pour k allant de 1 *à* $p-1$
        Calcul de $\mathcal{D}$
        On remplace la diagonale de $\mathcal{D}$ *par* ∞
        On trouve le minimum de $\mathcal{D}$
        On calcul la distance D_{LW} *entre ces deux clusters*
        On supprime de $\mathcal{D}$ *les deux colonnes et deux lignes*
        On ajoute la ligne et la colonne du nouveau cluster
        On met à jour P
    On renvoie P

2.2 Complexité temporelle

Comme énoncé au début de ce rapport, le grand problème de la classification ascendante hiérarchique est son coût de calcul qui s'avère particulièrement élevé, et notre algorithme ne va certainement pas y échapper, en effet, nous allons effectuer le calcul de la complexité temporelle de notre algorithme et en discuter.

Sans perte de généralités, nous ne nous attarderons pas sur certains aspects de l'algorithme qui sont trivialement "négligeables" comparé au coût total de l'algorithme. On comprend assez vite que la part importante du coût temporelle va être dû aux parcours de la matrice de dissimilarité $\mathcal{D}$ pour chaque itération afin de trouver le minimum de celle-ci. La matrice $\mathcal{D}$ étant symétrique, il suffit de chercher le minimum dans la partie triangulaire inférieure de la matrice, c'est-à-dire que pour une matrice de taille n, seulement $\frac{n(n-1)}{2}$ comparaisons sont nécessaires, ce qui est notre cas à chaque itération. Il faut aussi prendre en compte le fait que la matrice $\mathcal{D}$ est diminuée d'une taille à chaque itération. Comme énoncé précédemment, on suppose que le nombre de clusters p cherché est égale (ou sans perte de généralités, à peu près égale) à n.

Soit T la fonction de coût temporel dépendant de n le nombre d'individus et de d la dimension des données, on a :

$$
\begin{aligned}
T(n) &\approx d \times \frac{n(n-1)}{2} + \sum_{k=0}^{n-1} \frac{(n-k)(n-k-1)}{2} \\
&\approx d \times \frac{n(n-1)}{2} + \sum_{k=0}^{n-1} n^2 - n - 2nk + k^2 + k \\
&\approx d \times \frac{n(n-1)}{2} + n^3 - n^2 + \frac{n(n-1)(2n-1)}{6} - 2\frac{n^2(n-1)}{2} + \frac{n(n-1)}{2} \\
&\approx (\frac{d}{2} + \frac{1}{3})n^2 + n^3 + \frac{2n^3}{6} - n^3 + o(n^2) \\
&\approx \frac{1}{3}n^3 + (\frac{d}{2} + \frac{1}{3})n^2 + o(n^2)
\end{aligned}
$$

(le terme $d \times \frac{n(n-1)}{2}$ vient du calcul de la matrice $\mathcal{D}$ symétrique à la 1ère itération, car l'initialisation des distances de Lance-Williams nous impose de calculer les distances euclidiennes entre les individus dont la complexité est linéaire en d).

La complexité temporelle de notre algorithme est donc en $O(n^3 + dn^2)$, rendant ainsi compliquée le calcul sur des larges jeu de données, cela peut s'avérer extrêmement coûteux. Nous pouvons vérifier cela avec nos données en affichant le temps que met notre algorithme à tourner en fonction du nombre de données n, c'est ce que l'on voit sur le graphique suivant (d fixé, $d = 4$, method = single) :

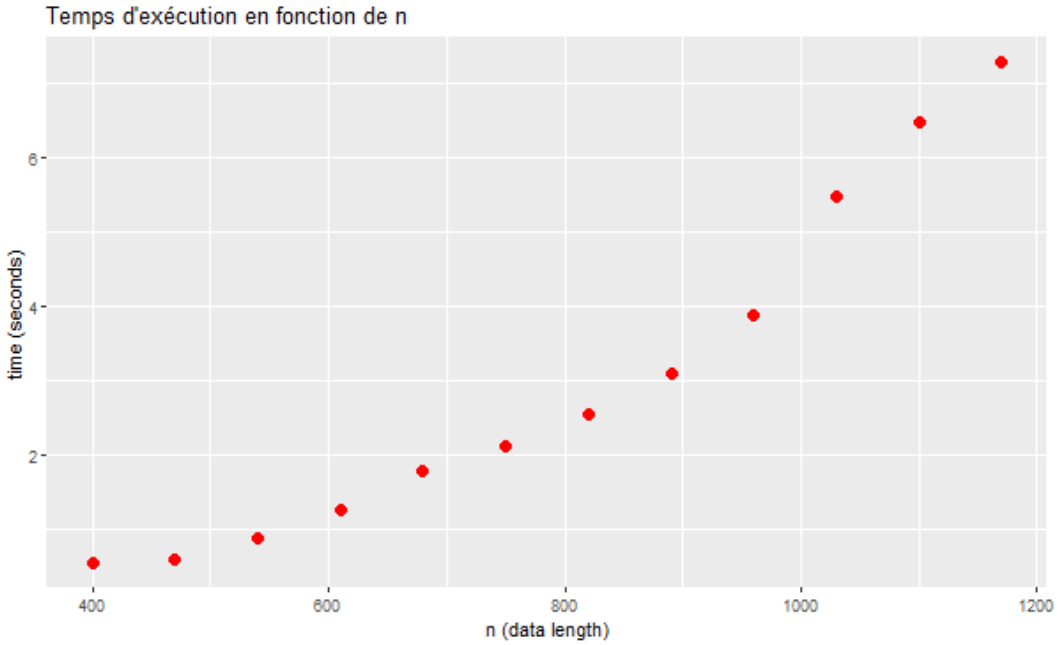


Figure 2: Coût temporel en fonction de n ($d = 4$)

Il est clair qu'il n'est pas évident de regarder sur un tel graphique si la courbe est bien cubique, cependant une manière simple de s'assurer du caractère cubique de notre complexité est simplement de prendre la racine cubique du temps d'exécution et de l'exposer en fonction de n, on retrouve bien alors un caractère linéaire de la courbe sur la figure [8] se trouvant en Annexe, prouvant le caractère cubique du coût temporelle. On peut aussi vérifier que pour un n fixé tel que $n^3 \approx n^2$ (c'est-à-dire pour n assez petit), alors le coût temporel est bien linéaire en la dimension d des données, en effet, sur le graphique suivant [9] (Annexe) on retrouve bien un caractère linéaire de notre complexité pour $d >> n$ (ici $n = 20$)

3 Algorithme optimisé : Nearest-Neighbor Chain

Dans cette partie, nous allons nous intéresser à un autre type d'algorithme pour résoudre le problème de classification hiérarchique ascendante. Il s'agit du Nearest-Neighbor Chain (ou **NNC**) pensé en 1983 par Murtagh et ses collègues [5] qui permet de réduire la complexité temporelle en $O(n^2)$. Nous présenterons d'abord son fonctionnement puis nous démontrerons sa complexité carrée

3.1 Heuristique

L'idée principale du NNC est que si deux clusters C_i et C_j sont plus proches voisins réciproques (ou **RNN**), alors ils seront fusionnés indépendamment des distances qui les séparent des autres clusters. On peut donc partir d'un cluster arbitraire et construire une chaîne de plus proches voisins (ou **NN**) jusqu'à atteindre un couple de RNN qu'on fusionnera. Les propriétés de ces chaînes de NN permettent de ne considérer que les autres clusters et l'avant dernier cluster de la chaîne pour la recherche d'un NN.

Néanmoins, la distance D_{LW} doit être réductible, c'est-à-dire que pour tout couple de RNN (C_i,C_j) et tout autre cluster C_k, elle doit vérifier l'inégalité :

$$D_{LW}(C_i \cup C_j, C_k) \geq \min(D_{LW}(C_i, C_k), D_{LW}(C_j, C_k))$$

Les distances réductibles présentes dans le tableau [1.1] sont celles des méthodes *single*, *complete* et *average*.

Sous condition de réductibilité, le NNC possède le même arbre et donc la même corrélation cophénétique que l'algorithme classique. La corrélation cophénétique est une mesure de dissimilarité entre deux résultats de clustering hiérarchique $k \in 1, 2$, définie comme la corrélation entre les distances cophénétiques $t_1(i, j)$ et $t_2(i, j)$ qui sont les hauteurs du premier noeud joignant les points i et j dans les dendrogrammes (par exemple dans celui-ci [1]).

Étant donné qu'il y a une part d'arbitrarité dans le choix du premier cluster d'une chaîne, l'ordre dans lequel les clusters seront fusionnés ne sera pas nécessairement le même que pour l'algorithme naïf. Ainsi, on doit prendre le nombre de clusters égale aux nombres d'individus (c'est-à-dire $p = n$) et résoudre le problème de classification jusqu'au bout pour que celui-ci soit correct. Nous pouvons résumer le fonctionnement de notre algorithme optimisé dans ce pseudo-code :

Algorithm 2 NNC

```
1: procedure
2:     On choisit un type de linkage
3:     On initialise la matrice des clusters P
4:     On initialise la matrice de dissimilarité 𝒟
5:     On initialise une pile vide
6:     Pour k allant de 1 à n − 1
7:         Si la pile est vide :
8:             On ajoute un cluster arbitraire à la pile
9:         Tant qu'on ne trouve pas de RNN :
10:            On ajoute à la pile le NN du dernier élément de la pile
11:        On fusionne le couple de RNN
12:        On les supprime de la pile
13:        On met à jour 𝒟
14:        On met à jour P
15:    On renvoie P
```

3.2 Complexité temporelle

Étant donné la dépendance importante entre le fonctionnement de l'algorithme NNC et les données utilisées, il est moins évident de calculer la complexité temporelle. On peut cependant s'intéresser à des cas limites afin de pouvoir la borner. Comme pour le cas classique, on ne s'intéressera ici qu'aux comparaisons qui étaient la cause de la complexité cubique del'algorithme classique.

Premièrement, prenons le cas trivial de n points unidimensionnel tels que $x_i = \frac{1}{2^i}$ traités avec la méthode *single*. Supposons que pour la première itération on choisisse x_1 comme premier élément de notre pile. La chaîne se construira jusqu'à x_n puisque x_n et x_{n-1} forment l'unique couple de RNN. Le couple de RNN suivant sera x_{n-2} et $x_n \cup x_{n-1}$ et ainsi de suite. Dans ce cas précis, à la première itération, on effectue $n - 1$ comparaisons lorsqu'on est en x_1, $n - 1$ en x_2, $n - 2$ en x_3, ...,1 en x_n. Ensuite, pour les itérations suivantes, on effectue 1 comparaison. Cela nous donne alors, pour T la complexité temporelle dépendant de n :

$$\begin{aligned} T(n) &\geq n - 1 + \sum_{i=2}^{n}(n - i + 1) + \sum_{k=2}^{n-1} 1 \\ &\geq (n - 1)(n + 3) - \frac{(n + 1)n}{2} + 1 \\ &\geq \frac{n^2}{2} + o(n^2) \end{aligned}$$

Il est cependant plus difficile de s'imaginer un jeu de données qui maximiserait la complexité. Dans l'idéal, la pile devrait rester petite pour avoir un grand nombre de comparaisons par NN recherché, mais elle devrait aussi s'agrandir pour que l'on recherche plusieurs NN par itération. Prenons un jeu de données hypothétique pour lequel on trouve systématiquement un RNN au bout du p-ème élément de la pile. On aura alors p recherches de NN pour toutes les itérations $k \leq n-p+1$, au-delà desquelles la pile sera remplie au maximum et on se retrouvera dans le cas trivial. Cela n'est pas gênant puisqu'il y aura peu de clusters à parcourir, sous condition que p soit assez faible. On a donc pour un p bien choisi :

$$\begin{aligned}
T(n) &\leq \sum_{k=1}^{n-p+1} \left(n-k+\sum_{i=0}^{p-2}(n-k-i) \right) + \sum_{k=n-p+2}^{n-1} 1 \\
&\leq p-3+\sum_{k=1}^{n-p+1} \left(n-k+(n-k)(p-1)-\frac{(p-1)(p-2)}{2} \right) \\
&\leq \left(1-\frac{(n-p+2)(n-p+1)}{2} \right) p + \left(np-\frac{(p-1)(p-2)}{2} \right)(n-p+1)-3 \\
&\leq \frac{p}{2}n^2+o(n^2)
\end{aligned}$$

On vient donc de démontrer par encadrement que $T(n) = O(n^2)$. Comme pour le cas classique, on a ici seulement considéré le temps mis pour rechercher et fusionner les clusters. La matrice $\mathcal{D}$ initiale est toujours calculée en $O(d \times n^2)$.

4 Évaluations des algorithmes

Dans cette dernière partie, nous allons présenter les protocoles d'évaluations de nos algorithmes ainsi que les résultats obtenus. Il sera question de confirmer et de quantifier les gains d'optimisations temporelle et spatiale. Comme mentionné précédemment, on utilisera les données iris comme jeu de données réelles. Nous simulerons également des jeux de données simples permettant ainsi de contrôler le nombres de points et la dimension.

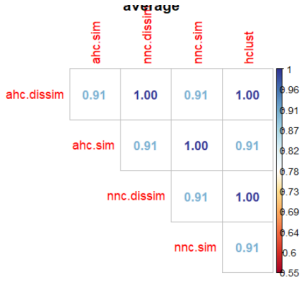
4.1 Dataset Iris

Tout d'abord, nous allons comparer nos 4 algorithmes (classique ou NNC avec ou sans usage des similarités) et celui du package hclust entre eux pour chaque méthode parmi celles du tableau [1.1]

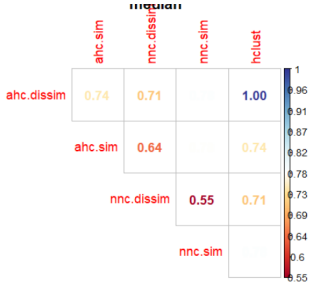
Au vu des valeurs des corrélations cophénétiques présentes dans la figure 3, on remarque que, pour les méthodes réductibles, les algorithmes classiques (préfixe « ahc ») donnent les mêmes résultats que les NNC. Néanmoins des différences plus ou moins importantes apparaissent lorsqu'on utilise une matrice de similarités, tout en conservant l'équivalence entre classique et NNC. On peut également voir que pour la méthode median, qui est non-réductible, nous n'avons pas d'équivalence comme attendu. L'algorithme du package hclust renvoie les mêmes résultats que l'algorithme naïf utilisant les dissimilarités.

Ensuite, nous nous intéressons au gain de complexité spatial dû au seuillage des similarités. Nous proposons de reproduire les résultats de l'article suivant [[1]] dont est tiré la méthode en appliquant aux données Iris centrées-réduites l'algorithme classique utilisant une mesure de similarités cosinus pour 20 valeurs de seuils avec un pas 0.05. On mesure à chaque fois le taux de similarités tombant sous le seuil (et donc non prises en compte dans la recherche des clusters les plus proches) et la corrélation cophénétique avec l'algorithme utilisant des dissimilarités.

Comme on peut le voir dans la figure 4, la corrélation cophénétique reste quasiment inchangées jusqu'à un seuil de 0.4. Pour cette valeur, on a déjà écrêté 40% de la matrice de similarités initiale, d'où l'utilité d'une telle méthode lorsque le nombre de données à stocker est légèrement trop grand (à un facteur inférieur à 10).



(a) average method



(b) median method

Figure 3: Comparaison méthodes average et median

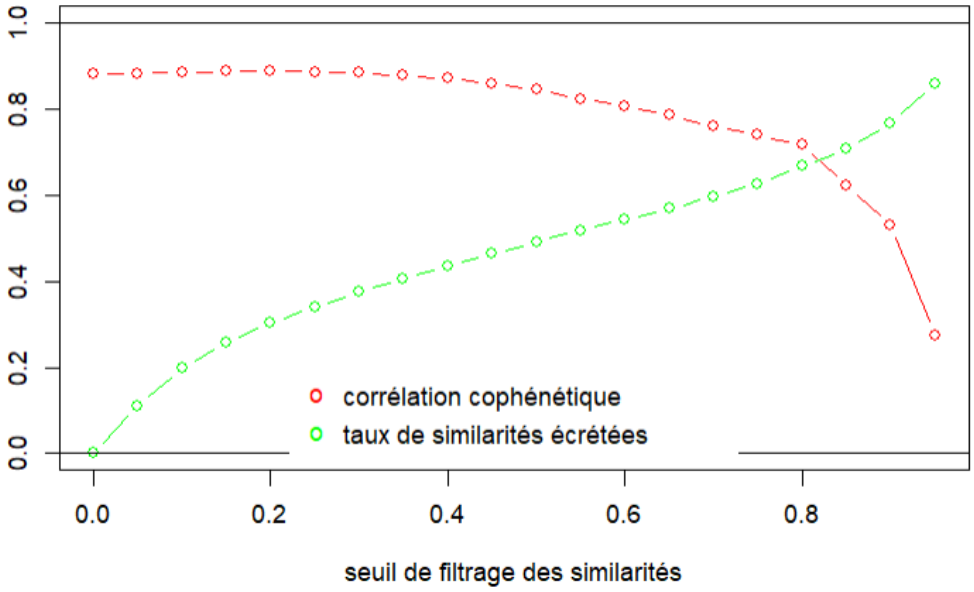


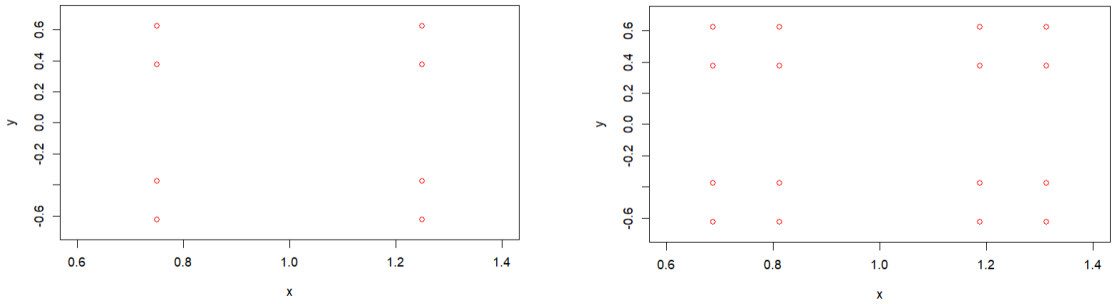
Figure 4: Seuil de filtrage des similarités

4.2 Données simulées

On s'intéresse désormais à la complexité temporelle de nos algorithmes en mesurant leurs temps d'exécution sur plusieurs jeux de données simulées de taille différente. Nous simulons d'abord 20 jeux de données « imbriquées » pour n variant de 128 à 2048. Nous choisissons $d = 2$ car les simulations et le calcul des matrices $\mathcal{D}$ et $\mathcal{S}$ seraient sinon trop coûteuses, empêchant ainsi de tester de grandes valeurs de n (ce qui nous intéresse). De plus, la complexité temporelle des algorithmes est indépendant de la dimension des données.

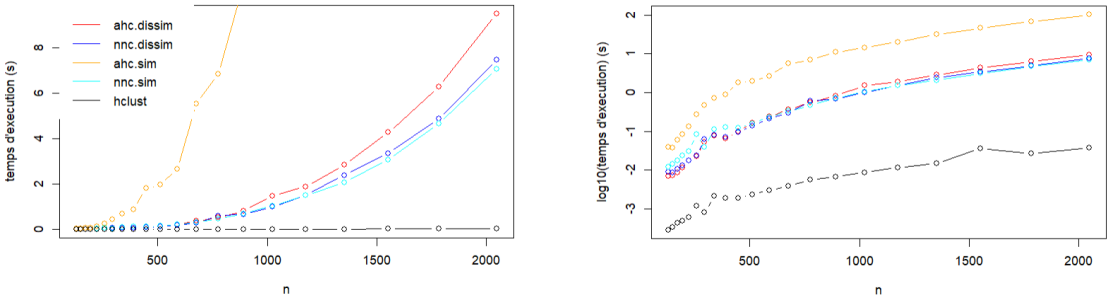
On peut visualiser sur la figure 5 la manière dont ses données sont simulées pour $n \in \{8, 16, 32\}$ et $d = 2$. Ce type de données présentent plusieurs avantages : il est facile de les produire, les classes sont triviales car imbriquées et la complexité théorique du NNC sur celles-ci se rapproche du pire cas.

On peut voir sur la figure 6 qu'on gagne en temps sur toutes les valeurs de $n \geq 500$ testées lorsqu'on utilise le NNC plutôt que le naïf. Par ailleurs, on remarque que pour le naïf utilisant $\mathcal{S}$, on a une complexité anormalement plus élevée. Cela est probablement dû aux calculs matriciels intervenant à chaque itération pour la recherche du minimum. Bien que le NNC soit plus optimal que le naïf, on a du mal à distinguer la réduction du $O(n^3)$ en $O(n^2)$ sur la plage de valeurs de n testées. Nous proposons donc de modéliser $T(n)$ pour les algorithmes utilisant $\mathcal{D}$ par un polynôme en n en utilisant la méthode des moindres carrés. Nous choisissons deux types de modèles, un de degré 2 et un de degré



(a) Simu 8 (b) Simu 16

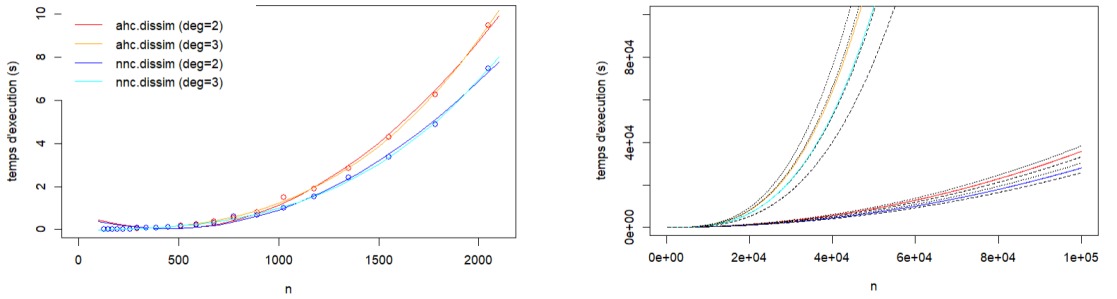
Figure 5: Simu 8 et Simu 16



(a) times 1 (b) times 2

Figure 6: times 1 et times 2

3. On applique des poids aux données intervenant dans le modèle de sortes à ce que les poids soit inversement proportionnels à la fréquence des valeurs disponibles au voisinage d'un n donné.



(a) prediction 1 (b) prediction 2

Figure 7: prediction 1 et 2

On récupère alors des courbes très similaires pour $n \leq 2048$, comme on peut le voir sur la figure 7. On observe par ailleurs que le modèle de degré 3 du NNC est très incertaine. En effet, son intervalle de confiance (en pointillées et traits coupés respectivement pour les bornes supérieure et inférieure) est très large, atteignant $[-10^4, +10^4]$ pour $n = 40000$. Cela va dans le sens de sa complexité théorique en $O(n^2)$. Néanmoins, on observe que le modèle de degré 2 du naïf est assez précis, ce qui n'est pas satisfaisant au vu de sa complexité théorique en $O(n^3)$. On peut expliquer cela par des valeurs de n testées trop faibles. En effet, comme on peut le voir sur les deux graphiques, les modèles de degré 2 et 3 ne commencent qu'à peine à diverger pour $n \geq 1000$. Des valeurs de n supérieures à 20000 auraient sûrement permis de distinguer les dépendance cubique et carrée de nos deux méthodes, mais

cela aurait probablement pris beaucoup trop de temps au vu des valeurs prédites. Toutefois, si l'on considère comme satisfaisantes les prédictions de degré 2 pour le NNC et de degré 3 pour le naïf, on remarque la très forte divergence des temps d'exécution et l'intérêt de la méthode optimisé.

5 Réduction de la complexité spatiale : seuillage des similarités

Dans la partie précédente, on a montré comment on pouvait réduire la complexité temporelle du clustering hiérarchique ascendant en $O(n^2)$. Ici, nous allons brièvement nous intéresser à une autre façon de concevoir les deux précédents algorithmes en utilisant les similarités plutôt que les dissimilarités (ou distances). Comme on peut le voir dans l'article [[1]], des similarités entre plusieurs clusters peuvent être reformulées pour donner les dissimilarités sur lesquelles se basent la recherche des clusters à fusionner. La méthode proposée dans cet article vise à utiliser une matrice initiale de similarités $\mathcal{S}$ plutôt que $\mathcal{D}$. Cela permet de seuiller la matrice et de la rendre éparse et donc moins coûteuse en mémoire.

6 Conclusion

Nous avons dans ce rapport fait une revue des méthodes et concepts courants de la classification ascendante hiérarchique et avons présenté des algorithmes permettant de réduire le coût spatial et temporel de cette tâche. De plus, nous avons confirmé par des expériences sur données réelles et simulées l'avantage de tels algorithmes. Bien que les dépendances cubique et carrée de nos algorithmes n'aient pas été clairement prouvées par la pratique, nous pourrions avec de meilleurs moyens matériels et des expériences plus longues aisément en faire les démonstrations.

7 Annexe

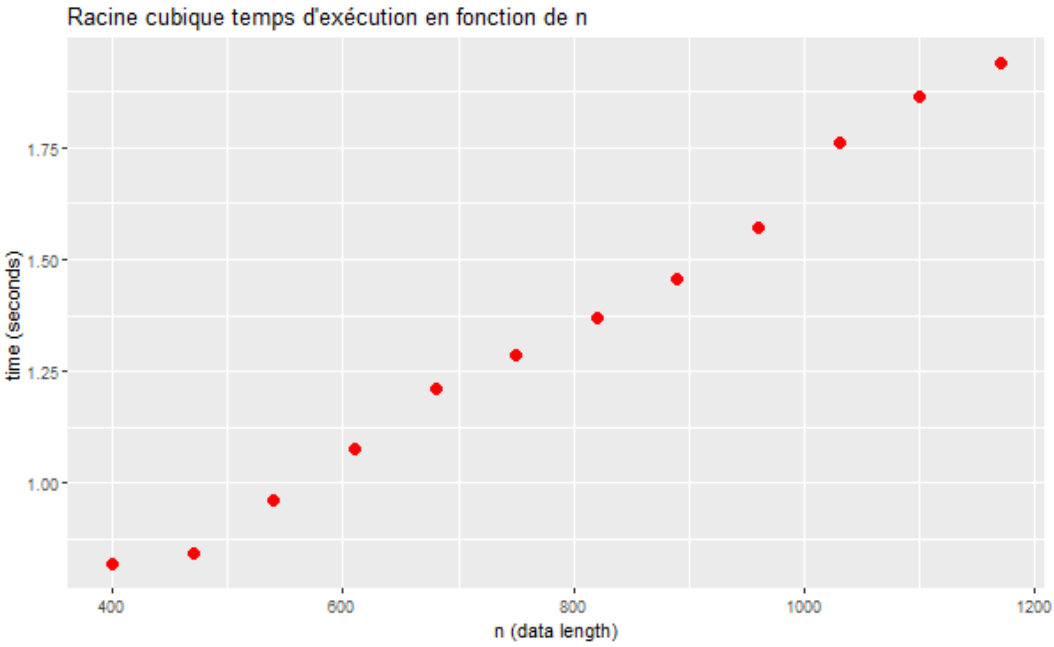


Figure 8: Racine cubique du coût temporel en fonction de n ($d = 4$)

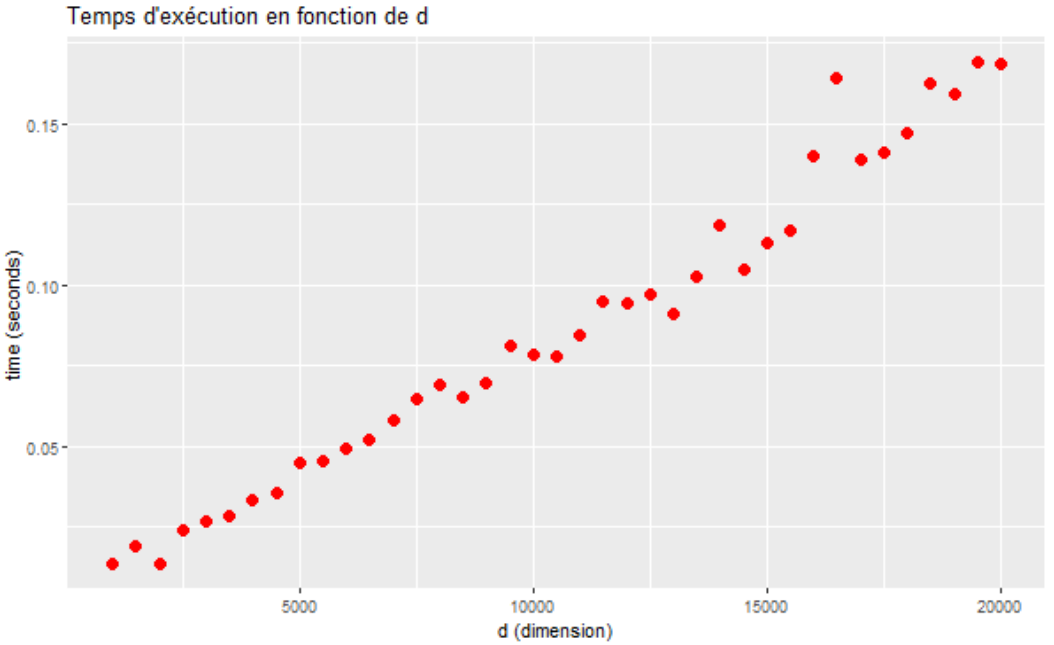


Figure 9: Coût temporel en fonction de d ($n = 20$)

References

[1] Julien Ah-Pine and Xinyu Wang. Classification ascendante hiérarchique à noyaux et pistes pour un meilleur passage à l'échelle. In *Journées de Statistique de la SFDS*, 2015.

[2] Sergiu Theodor Chelcea. *Agglomerative 2-3 Hierarchical Classification: Theoretical and Applicative Study*. PhD thesis, Université Nice Sophia Antipolis, 2007.

[3] Mohammad Ghasemigol, Mostafa Sabzekar, Reza Monsefi, and Hadi sadoghi yazdi. A new tree clustering algorithm for fuzzy data based on alpha-cuts. volume 1, pages 210–216, 07 2009.

[4] T Soni Madhulatha. An overview on clustering methods. *arXiv preprint arXiv:1205.1117*, 2012.

[5] Fionn Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The computer journal*, 26(4):354–359, 1983.

[6] MA Syakur, BK Khotimah, EMS Rochman, and Budi Dwi Satoto. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP conference series: materials science and engineering*, volume 336, page 012017. IOP Publishing, 2018.

[7] Gavriel Yarmish, Philip Listowsky, and Simon Dexter. Distributed lance-william clustering algorithm. *arXiv preprint arXiv:1709.06816*, 2017.