Siméon GAUMART

Nguyet Ha TRAN

5SDBD - B1

**Rapport TP Clustering**

# Introduction

De nos jours, l’apprentissage est un domaine important des sciences informatiques qui permet aux ordinateurs d'apprendre sans être programmés. Les deux principales catégories de l'apprentissage utilisées sont apprentissage supervisé et apprentissage non supervisé. En fait, l'apprentissage non supervisé est un domaine beaucoup plus difficile de l'apprentissage, où nous apprenons à partir de données non labellisées au lieu de données labellisées dans le cas de l'apprentissage supervisé. Dans ce travail pratique, nous nous concentrons principalement sur l'exploration de différents algorithmes de clustering.

Ce rapport a pour but de rendre compte de nos travaux en TP d’apprentissage non supervisé. Le code peut être trouvé [ici](https://colab.research.google.com/drive/1x1_nAFgZ2oYdt1EMpJmSfWd_JCZP53z7?usp=sharing).

# Jeux de données

Pour ce TP, nous avons utilisé des jeux de données artificielle disponible sur le github <https://github.com/deric/clustering-benchmark>.

Nous avons récupéré les coordonnées x, y et les labels de chaque fichier .arff dans des dictionnaires comportant les noms de fichier pour accéder aux bonnes données.

Pour la partie synthèse, nous avons transformé les fichiers textes contenant les données en fichier CSV (avec virgule) grâce à un tableur. Nous avons ensuite récupéré les données grâce à la librairie pandas et avons stocké chaque données de fichier dans une matrice, le tout regroupé dans un dictionnaire accessible avec le nom de fichier.

# Méthodes implémentées

## Clustering *k*-Means

Nous commençons notre TP de clustering avec K-Means Clustering. Il s'agit d'une méthode d'apprentissage non supervisée très populaire, grâce à sa mise en œuvre simple, sa méthodologie simple à comprendre et directe qui fournit la plupart du temps des performances robustes et efficaces, notamment pour le partitionnement de la plupart des ensembles de données avec des attributs numériques. Son idée est le clustering basé sur les centroïdes, qui considèrent le centre des points de données comme le centre du cluster correspondant, en cherchant à partitionner l'ensemble de données en clusters égaux.

### Qu’est-ce que c’est la méthode de clustering *k*-Means et quand l’utiliser?

L’idée principale de la méthode *k*-Means est : pour un nombre k prédéfini de clusters, l’algorithme met à jour itérativement les k centroïdes et affecte à chaque échantillon le plus proche centroïdes afin de répondre aux critères suivants:

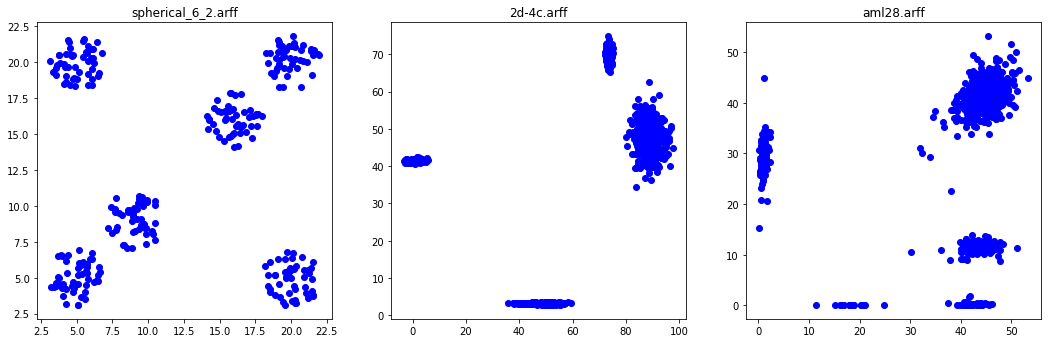
* La distance inter-cluster (la distance entre chaque échantillon d’un cluster et son centroïdes) est réduite au minimum afin de garantir la propriété cohésion des clusters
* La distance intra-cluster (la distance entre 2 clusters) est maximisée, de sorte que les clusters sont bien séparés.

après quelques itérations, l’algorithme trouve un découpage stage du jeu de données (convergence)

L’algorithme choisit les k centroïdes aléatoirement ou de façon “k-means++” où les centroïdes sont initialisés loin entre eux. Ensuit, il déplace itérativement les centroïdes vers les centres des échantillons pour but de minimiser l’inertie ( la somme des carrés inter-clusters) jusqu’à ce qu’il atteigne la stabilisation de l’inertie totale, c’est-à-dire les centroïdes ne bouge pas significativement.

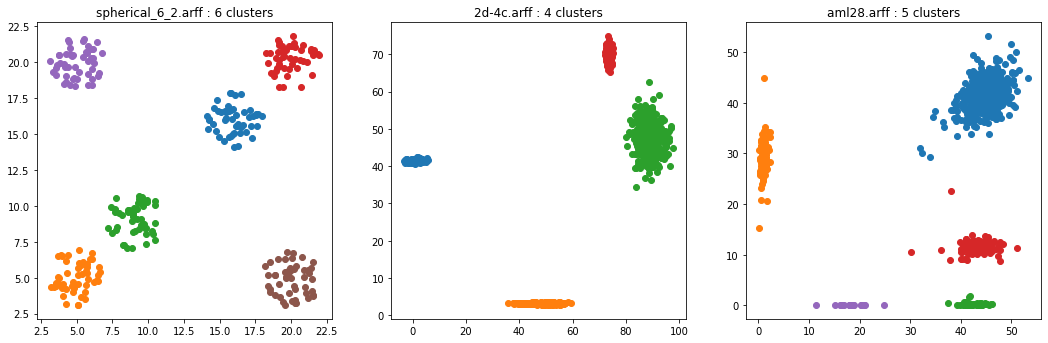
On peut observer que cet algorithme optimise les centroïdes des clusters au lieu de leurs frontières. De plus, la métrique de distance utilisée dans l’algorithme *k*-means est la distance euclidienne, qui lui permet d’identifier très bien les clusters dans les jeux de données dont les échantillons sont représentés dans la grille 2D. Donc, cet algorithme préfère les clusters de tailles similaires (1). En outre, d’après scikitlearn, l’inertie suppose que les clusters sont convexes et isotropes, l’algorithme pourrait donc ne pas fonctionner correctement avec des clusters de formes irrégulières (par exemple : spiral) (2). En plus, même si le jeu de données ne satisfait pas aux précédentes conditions, l’algorithme *K*-meanspeut toujours fonctionner avec un jeu de données si leurs clusters sont bien séparés grâce à “k-means++” qui initialise les centroïdes très éloignés les uns entre les autres (3)

Pour toutes ces raisons (1,2 et 3), nous avons choisi les 3 jeux de données (présentés dans la figure 1) pour lesquels il nous semble que la méthode *k*-Means pourrait identifier les clusters correctement



**Figure 1**: Représentations les 3 jeux de données choisis pour la méthode *K*-Means

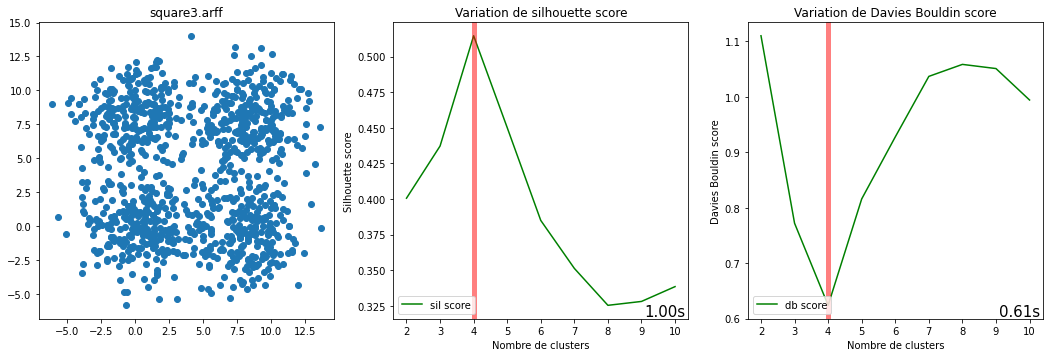
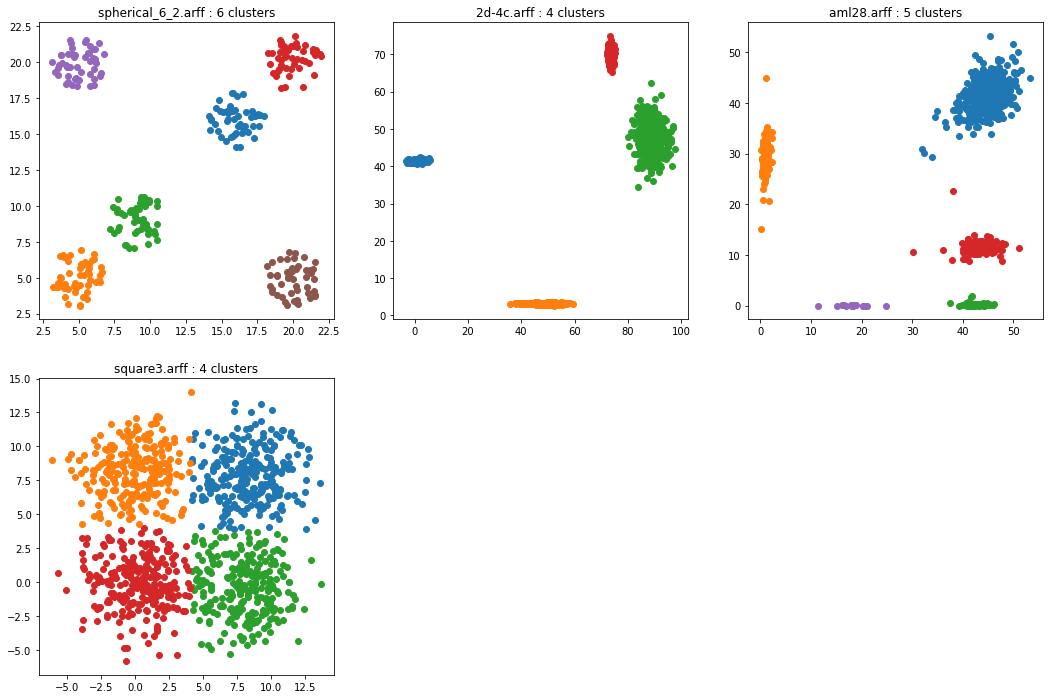
Nous allons donc utiliser la méthode *K*-Means afin de voir les résultats obtenus. Les résultats obtenus sont présentés dans la figure 2.



**Figure 2**: Les résultats obtenus en appliquer la méthode k-Means sur les jeux de données choisis

### Comment paramétrer pour mieux appliquer la méthode de clustering *k*-Means

Le fait que la méthode *k*-Means n’a qu’un seul paramètre important: le nombre de clusters, explique pourquoi il est relativement simple à implémenter. Mais dans certains cas, le choix de la valeur k peut être ne pas évident, il sera donc nécessaire de vérifier la qualité du clustering. Nous présenterons ici deux méthodes que l’on a utilisées pour calculer le score de clustering et par conséquent, ces méthodes nous aident à choisir une valeur k optimale.

Nous avons exploré les paramètres de la méthode *k*-Means sur le jeu de données “square3.aff”. Par observation, nous pouvons dire que ce jeu de données contient 4 clusters. Mais 4 est-il le nombre de clusters optimal?

**Figure 3**: Représentation originale du jeu de données, variation de son score de silhouette et son score de Davies Bouldin en fonction de k et son résultat obtenues avec la méthode *k*-Means

La première façon que l’on a choisi est de calculer et comparer le score de silhouette pour différentes valeurs de k de 2 à 10. La figure 3 ci-dessus montre que 4 est la valeur k optimale, compte tenu de son plus élevé score de silhouette. Pour la deuxième façon, on a appliqué le même principe pour comparer le score de Davies Bouldin. Aussi sur la figure 3, on observe que 4 est la valeur k optimale, compte tenu de son plus petit score de Davies Bouldin.

### Limites de la méthode

Ensuite, nous appliquons la méthode *k*-Means aux jeux de données pour lesquels il nous semble que la méthode *k*-Means ne pourrait pas identifier les clusters correctement.

Pour toutes ces raisons citées dessus (1,2 et 3), nous avons choisi ces 3 jeux de données (présentés dans la figure 4) car soit leurs clusters ont des formes irrégulières, soit leurs centroïdes ne sont pas bien éloignés les uns entre les autres, soit les tailles de leurs clusters ne sont pas similaires.



**Figure 4**: Les jeux de données montrant des limites de la méthode de clustering *k*-Means

D’après le travail qu’on a effectué et l’analyse dans les sections précédentes, nous avons tout d'abord observé que, grâce à sa façon d'optimisation des centres de gravité, la méthode de clustering *k*-Means a une vitesse de calcul assez rapide et le potentiel d’extension vers des plus grands jeux de données. Elle est également simple à implémenter, puisqu'il suffit de fournir le nombre de clusters et le métrique de distances (si nécessaire). Pourtant, un seul paramètre principal est aussi considéré comme un inconvénient: dans plusieurs situations réelles, le nombre de clusters n’est pas évident,de sorte qu'il peut être nécessaire de faire des efforts pour trouver la valeur K optimale et un mauvais choix de valeur de k donnera un mauvais résultat. En outre, comme les jeux de données réels contiennent souvent le bruit d’échantillonnage et que l’algorithme calcule itérativement les centroïdes des clusters, le bruit sera éventuellement pris en compte dans les calculs et donc, donnera un résultat final incorrect. On peut l’observer dans l’exemple du jeu de données “cluto-t4-8k.arff” (Figure 4). De plus, comme la valeur de k a un impact direct sur la complexité de l'algorithme, cette méthode ne conviendra pas au jeu données comportant dont le nombre de clusters est élevé.

En outre, l'algorithme indique également que les centroïdes de départ doivent être initialisés de manière aléatoire, de sorte que la stabilité des performances n'est pas garantie, par conséquent, parfois, l'algorithme pourrait converger vers un optimum local. De plus, comme on a mentionné dessus, cette méthode ne peut pas être utilisée pour les jeux de données dont les tailles des clusters ne sont pas égales ou dont la densité se varie.

## Clustering agglomératif

Dans cette partie nous allons utiliser une méthode de clustering agglomératif.

Le clustering marche de la façon suivante : Au départ chaque points du jeu de données est considéré comme étant un cluster. On lie ensuite les clusters entre eux (on les fusionne) en suivant une politique de liage. Par exemple, on peut fusionner les deux clusters dont les centres sont les plus proches (en distance euclidienne). On fusionnera donc les deux clusters les plus proches à chaque itération jusqu’à avoir le nombre de clusters voulu.

Sur Python, avec scikit learn, on utilise la méthode Agglomerative Clustering pour performer l’algorithme de clustering agglomératif sur les jeux de données. Cette méthode permet de choisir le nombre de clusters voulu (n\_clusters) et la méthode de liage des clusters (linkage), ainsi que d’autres paramètres qui ne nous intéressent pas ici. Dans les méthodes de liage possible nous avons :

* ***ward***, qui permet de fusionner les deux clusters dont la variance (après fusion) est la plus faible.
* ***average***, qui permet de fusionner les deux clusters dont la moyenne de la distance entre chaque point de deux clusters est la plus petite.
* ***complete***, qui permet de fusionner les deux clusters dont la distance entre les deux points les plus éloignés des deux clusters est la plus petite.
* ***single***, qui permet de fusionner les deux clusters dont la distance entre les deux points les plus proches des deux clusters est la plus petite.

Nous allons essayer la méthode de clustering agglomératif sur 3 jeux de données :

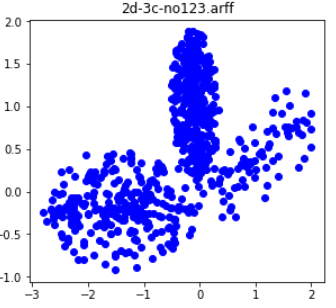
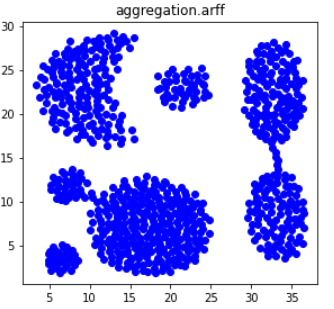
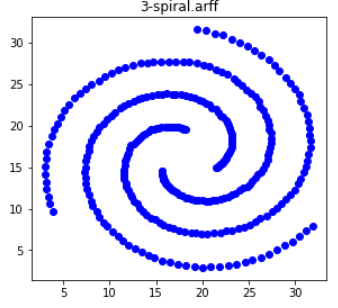


Figure 5

Nous allons donc utiliser la méthode de clustering agglomératif avec les différentes méthodes de liage afin de voir les résultats obtenus :

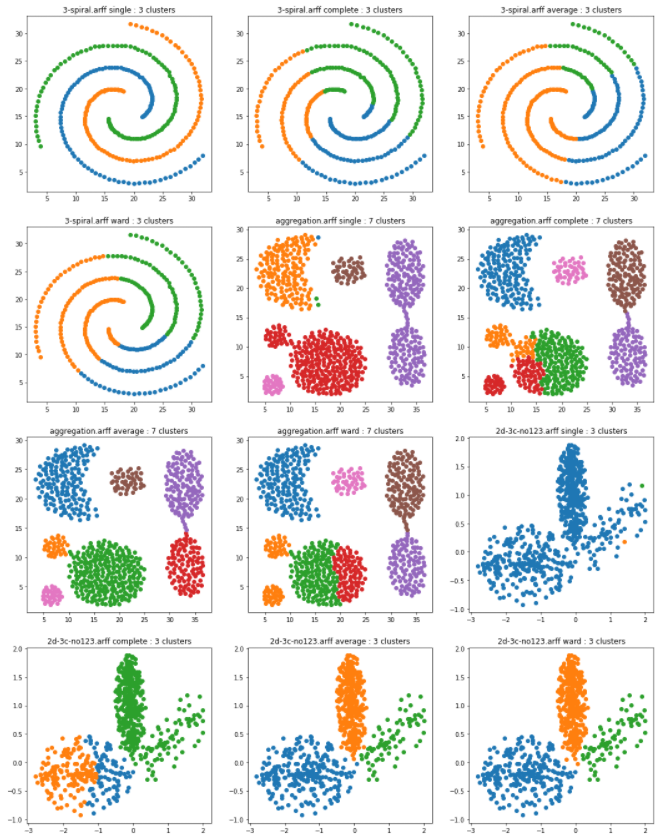


Figure 6

On remarque que selon les jeux de données, les différentes méthodes de liage ne donnent pas les mêmes résultats et donnent plus ou moins des résultats satisfaisants.

Pour 3-spiral.arff, on remarque que seulement la méthode de liage single donne le bon résultat. En effet, les spirales étant assez proche les unes des autres, le fait d’utiliser la moyenne des distances, la distance maximale et la variance finira par fusionner des bouts de spirale, alors qu’en utilisant la distance minimale, les deux clusters fusionnés appartiendront toujours à la même spirale.

Pour aggregation.arff, on remarque que seulement la méthode de liage average donne le bon résultat. En effet ce jeu de données comporte des clusters bien regroupés et la méthode de liage average est parfaite pour ce genre de cas vu qu’elle prend la moyenne des distances entre tous les points des deux clusters (et donc les résultats de cette méthode se rapprochent de ceux de k-means car prendre la moyenne revient à prendre la distance entre les deux centroïdes des deux clusters). La méthode de liage single ne fonctionne pas bien car les clusters ne sont pas tous séparés distinctement et donc la méthode va regrouper les clusters n’étant pas bien séparés. La méthode de liage complete ne fonctionne pas bien car on a dans ce jeu de données un cluster assez large proche de deux petits clusters, et donc une partie du cluster assez large va être regroupée avec les petits clusters car la distance entre les points gauche du gros cluster et les points les plus éloignés des petits est plus petite que la distance entre les points gauche et droite du gros cluster. La méthode de liage ward ne fonctionne pas bien non plus un peu pour les mêmes raisons que précédemment, ayant deux petits clusters proche d’un gros, la variance des deux petits regroupés va être inférieure à celle du gros, et ils vont donc être regroupés, entraînant la division du gros (car on a demandé 7 clusters).

Pour “2d-3c-no123.arff”, on remarque que les méthodes de liage average et ward donnent le bon résultat car les points des clusters sont quand même assez bien regroupés (on arrive plutôt bien à les distinguer) et le degré de dispersion inter-cluster est plus faible que le degré extra-cluster (des clusters entre eux). L’algorithme ne marche cependant pas bien avec la méthode de liage single car la densité de points fluctue trop dans le jeu de données et les clusters sont collés entre eux. La méthode complete ne fonctionne pas non plus pour la même raison que le jeu d’avant : un des clusters est trop large par rapport aux autres.

Nous avons ensuite utilisé le score silhouette et le score Davies Bouldin pour déterminer automatiquement le nombre de clusters du jeu de données aggregation.arff. Pour cela on cherche pour quel nombre de cluster (en itérant) le score silhouette est le plus élevé (et le moins élevé pour Davies Bouldin), en utilisant la méthode de clustering agglomératif avec la méthode de liage average (qui marche bien sur ce jeu de donnée).

On obtient les résultats suivant :

silhouette\_score

{'cityblock': [4, 0.15645956993103027], 'cosine': [3, 0.26238274574279785], 'euclidean': [4, 0.2858459949493408], 'haversine': [2, 1.196627140045166], 'l1': [4, 0.14843034744262695], 'l2': [4, 0.2824060916900635], 'manhattan': [4, 0.19267535209655762], 'nan\_euclidean': [4, 0.42562317848205566]}

davies\_bouldin\_score

{'euclidean': [7, 0.09754586219787598]}

(sous la forme {metric: [nombre de clusters, temps d'exécution en s]})

Sachant que le véritable nombre de clusters du jeu de données aggregation.arff est 7, on remarque donc que le score silhouette n’est pas très approprié pour la méthode de clustering agglomératif avec ce jeu de données et avec n’importe quel metric. Ce n’est cependant pas le cas du score Davies Bouldin qui renvoie le bon nombre de clusters.

La méthode de clustering agglomératif ne marche pas sur n’importe quel jeu de données. Il y a des jeux de données où la méthode ne marche pas peu importe la méthode liage.

Nous allons montrer un exemple avec les jeux de données :

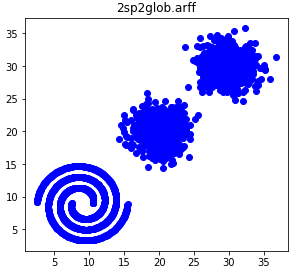
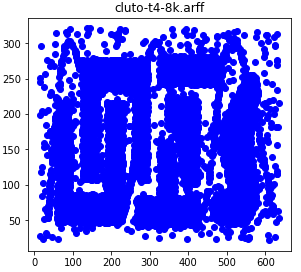


Figure 7

La méthode de clustering agglomératif donne comme résultats avec les différentes méthodes de liage :

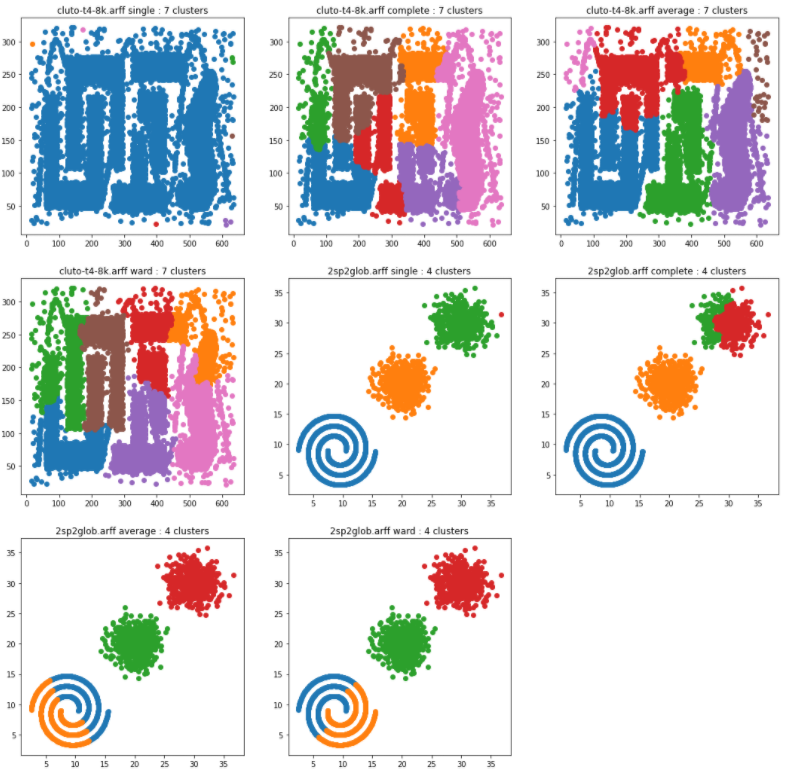


Figure 8

On remarque qu'aucune méthode de clustering agglomératif (aucun liage) ne fonctionne correctement pour ces deux jeux de données.

En effet, pour "cluto-t4-8k.arff'', les clusters sont collés les uns aux autres, la densité de points du jeu de données varie et les clusters s’entremêlent entre eux. Aucune méthode de liage disponible ne peut donc donner un résultat satisfaisant.

Pour “2sp2glob.arff” on a une combinaison de spirale et de cluster groupés. Nous avons vu sur les exemples d’avant que ce n’était pas les mêmes méthodes de liage qui donnaient de bons résultats pour les spirales et les clusters groupés. De plus, même la méthode de liage single ne peut pas marcher avec la spirale de ce jeu de données car les points des deux différentes spirales sont plus proches que certains points d’un même cluster groupé. On aura donc un point seul qui formera un cluster et les deux spirales regroupées qui en formeront un autre.

## Clustering DBSCAN

Dans cette partie nous allons utiliser la méthode de clustering DBSCAN.

DBSCAN marche de la façon suivante : pour chaque points du jeu de données nous allons dessiner un cercle dont le centre est le point et de rayon “**eps**” (un paramètre à donner dans la méthode DBSCAN). Si le nombre de points dans le cercle dessiné est supérieur ou égal à “**min\_samples**” (un autre paramètre de la méthode DBSCAN à donner), tous les points et le point centre du cercle font partie du même cluster. Sinon, le point centre du cercle est considéré comme étant du bruit, sauf si il est dans un cercle d’un autre point ayant suffisamment de points dans son cercle (et fera donc partie de son cluster). Pour que deux points soient dans un même cluster, il faut donc pouvoir créer un chemin pour passer de l’un à l’autre tout en restant dans le même cluster.

Sur python, avec scikit learn, on utilise la fonction DBSCAN, où on peut choisir le rayon du cercle eps, et le nombre de points minimum du cercle (pour être dans le cluster) “**min\_samples**”.

Nous allons essayer DBSCAN sur deux jeux de données (les mêmes qui n’avaient pas marché avec la méthode de clustering agglomératif):

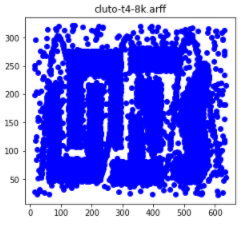
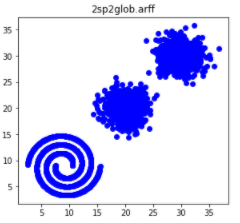
 

Figure 9

Pour les paramètres “**eps**” et “**min\_samples**”, on essaye d’observer les distances approximatives des points dans les clusters, afin de choisir des valeurs cohérentes.

Pour “cluto-t4-8k.arff” on choisit “**eps**” = 5.2 et “**min\_samples**” = 4, et pour “2sp2glob.arff” on prend “**eps**” = 1.5 et “**min\_samples**” = 10.

On obtient les résultats suivants :

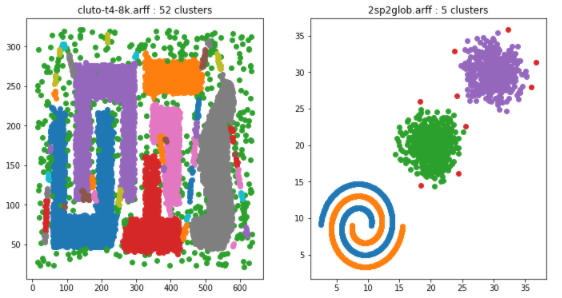


Figure 10

On remarque que DBSCAN arrive bien à identifier les clusters sur ces deux jeux de données, même si le résultat n’est pas parfait. Pour “cluto-t4-8k.arff”, avec les paramètres choisis, certains points étant du bruit sont regroupés comme étant un cluster car ils sont assez proche pour être considéré comme tel.

On essaye donc d’améliorer le résultat de “cluto-t4-8k.arff” en trouvant des paramètres plus adaptés.

Pour cela, on va itérer sur eps et “**min\_samples**” tout en testant le score de DBSCAN par rapport au résultat réel en utilisant “**adjusted\_rand\_score**” (pour un cas réel on ne connaît presque jamais le résultat réel, et donc ce ne serait pas possible d’utiliser adjusted\_rand\_score. On pourrait donc utiliser le score silhouette ou Davies Bouldin qui permettraient aussi de trouver les paramètres, mais avec un résultat moins satisfaisant).

On trouve donc que le meilleur score possible (avec les paramètres d’itération choisis) est d’environ 97,5%, et que les paramètres de DBSCAN pour ce score sont “**eps**” = 8.5 et “**min\_samples**” = 15.

En effet on remarque, même visuellement, que ces paramètres permettent de trouver un résultat plus satisfaisant :

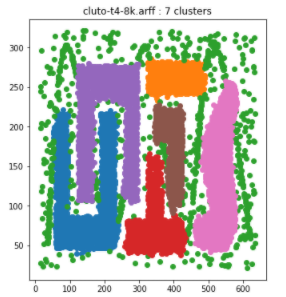


Figure 11

Cependant, la méthode DBSCAN ne fonctionne pas sur tous les jeux de données. En effet, si la densité de points varie dans le jeu de données (que ce soit entre les différents clusters ou dans un même cluster), la méthode DBSCAN ne sera pas efficace sachant que les paramètres eps et min\_samples restent les mêmes sur un même jeu de données. On aura donc des clusters qui seront fusionnés, des clusters qui apparaîtront comme étant du bruit, ou des sous-clusters qui se forment à l'intérieur d’un même cluster. La méthode DBSCAN n’est pas efficace non plus pour les jeux de données où les clusters ont la même densité mais sont collés les uns aux autres. Dans ce cas, les clusters seront tous fusionnés en un seul cluster.

Nous avons essayé la méthode DBSCAN sur un jeu de données où on peut retrouver ces problèmes :

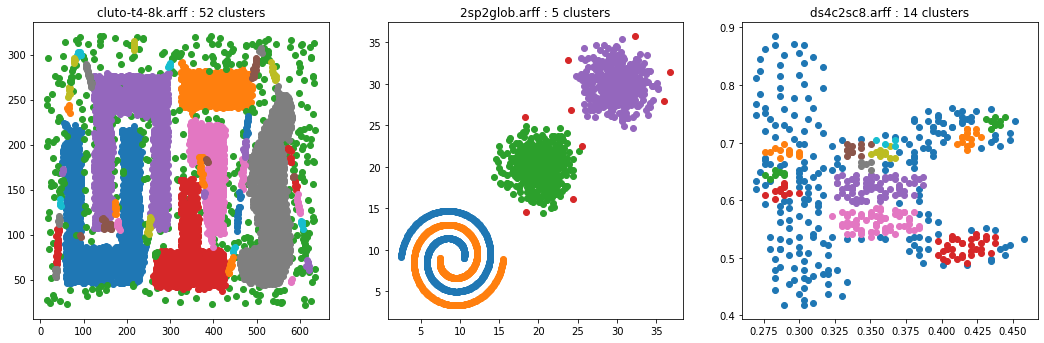


Figure 12

Le jeu de données ayant des clusters collés, avec une densité qui varie de façon assez aléatoire, la méthode DBSCAN n’est pas capable de donner des résultats satisfaisants quel que soit les paramètres donnés.

## Clustering HDBSCAN

À cause des limites de la méthode DBSCAN en traitant les jeux de données dont les densités des clusters sont inégales, les spécialistes ont mis en œuvre une nouvelle méthode qui s’appelle “Hierarchical Density-Based Spatial Clustering of Applications with Noise” (HDBSCAN). Cette méthode qui est une extension de la méthode DBSCAN permet de mieux partitionner des jeux de données dont la densité des clusters varie. Il améliore la méthode DBSCAN en le convertissant en un algorithme de clustering hiérarchique, puis en extractant un clustering plat basé sur la stabilité des clusters.

Dans cette méthode, une autre métrique de distance est utilisée pour distinguer les bruits l’un de l’autre et des clusters - la distance d’accessibilité mutuelle qui est défini comme tel : la distance d'accessibilité mutuelle entre deux points A et B est le maximum entre la distance centrale de A, la distance centrale de B et la distance entre A et B. Alors, on peut observer que plus le paramètre “***min\_samples***” est grand, plus de points sont considérés comme bruit.

L’étape suivante consiste à appliquer cette nouvelle métrique de distance au jeu de données. Puis, on construit un arbre couvrant minimal pour la distance d'accessibilité mutuelle récemment calculée, qui ne relie que les points dont la distance d'accessibilité mutuelle est supérieure à un certain seuil. Avec l’arbre couvrant, on peut ensuite le convertir en une hiérarchie des composants connectés, en triant les bords de l’arbre pas distance (dans l’ordre croissant), puis en les répétant et en créant un nouveau cluster fusionné pour chaque bord. Comme cet arbre contient encore une hiérarchie de clusters très large et compliquée, il doit être condensé en un arbre plus petit avec un peu plus de données attachées à chaque nœud. Un tel arbre condensé ressemble aux clusters dont la densité décroissante sur une distance variable est représentée sous la forme d'un triangle inversé. On utilise ensuite un autre paramètre important de HDBSCAN - “***min\_cluster\_size***”. Avec ce paramètre, on peut maintenant parcourir la hiérarchie, séparer les clusters qui n'atteignent pas cette taille minimale (le cluster le plus grand contient l’identité du cluster parent) ou garder ceux qui satisfont la taille dans l’arbre. Enfin, pour extraire des clusters, on veut sélectionner des clusters de l’arbre qui ont la plus grande surface de densité totale, c’est-à-dire, les points qui n'appartiennent pas à cette surface sont considérés comme bruit.

Par les principes expliqués, nous pouvons dire que la méthode HDBSCAN cherche à améliorer les limites de la méthode DBSCAN qui ne permet pas d’identifier des clusters dont les densités sont inégales. Cette méthode consistant à appliquer le clustering hiérarchique au clustering basée sur la densité rend la densité globale du jeu de données plus flexible.

Afin de vérifier les avantages de la méthode HDBSCAN, on l’utilise avec les deux jeux de données “2d-4c-no4” et “cluto-t7-10k”.

Concernant les paramètres, “**min\_cluster\_size**” et “**min\_samples**” sont les 2 paramètres les plus importants, “**min\_cluster\_size**” définit le plus petit taille de regroupement que nous souhaitons considérer comme un cluster, et “**min\_samples**” fournit le nombre de points minimum besoin pour former un cluster, plus la valeur “**min\_samples**” est grande, plus il y a de points considérés comme bruit, et les grappes seront limitées à des zones plus denses.

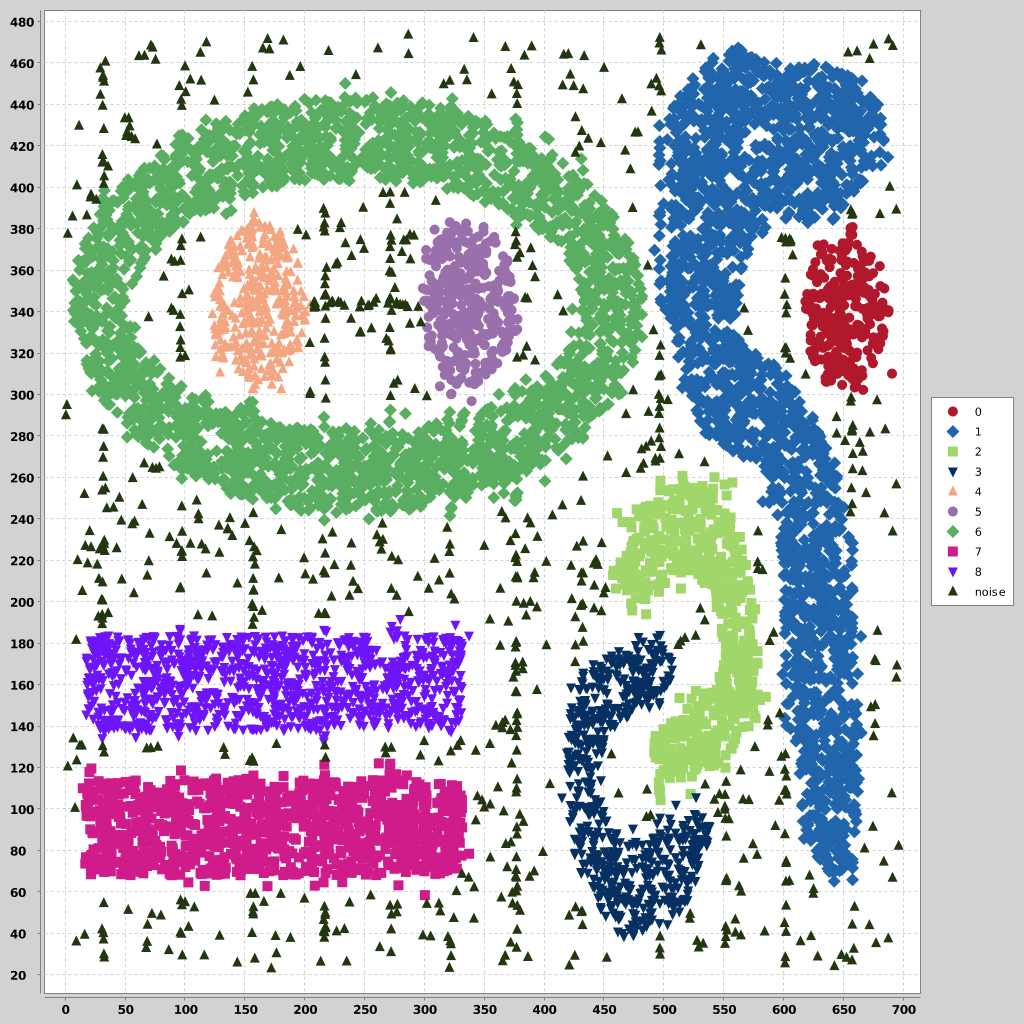


Figure 13: Résultat attendu[[1]](#footnote-0)

Pour le jeu de donées “cluto-t7-10k”, on a appliqué la même procédure que la méthode présentée dans la partie DBSCAN pour trouver que, “**min\_samples**” égal à 15 semble une bonne valeur. Pour paramétrer la valeur “min\_cluster\_size”, on l’a fait varier dans l’intervalle de 200 à 450



En comparant avec le résultat attendu, on observe que pour “**min\_cluster\_size**” = 200, le résultat obtenu montre que l’algorithme a bien détecté les 3 clusters en haut à gauche et les 2 à droite, pourtant, les restes restent incorrects.

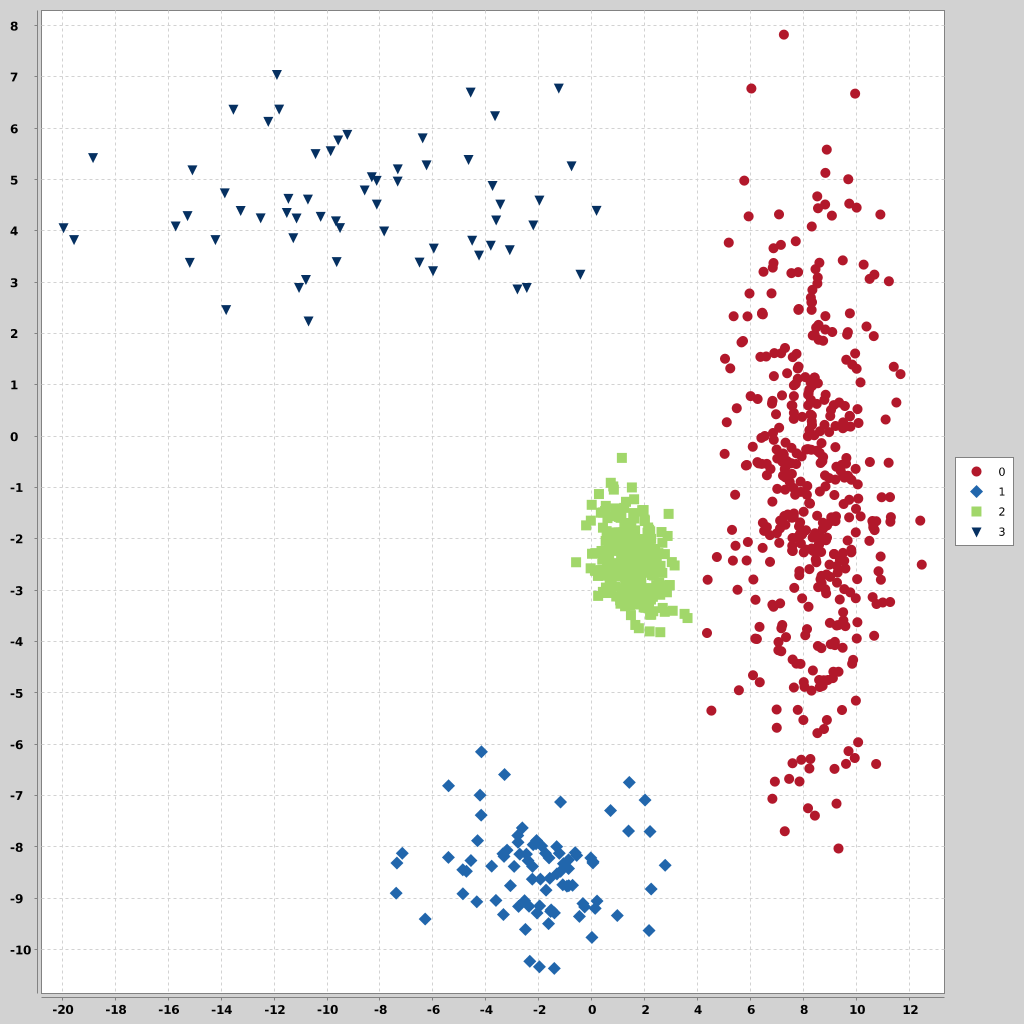


Figure 14: Résultat attendu[[2]](#footnote-1)

Ensuite, nous avons choisi le jeu de donné “2d-4c-no4” pour tester la méthode de DBSCAN. Les jeux de données similaires à celui-ci sont des “faiblesses” de la méthode DBSCAN à cause des densités de leurs clusters. Le jeu de données que l’on utilise ici est assez simple, ne contient pas de bruit. Dans cette partie, nous fixons donc la taille “**min\_cluster\_size**” à 12 et nous faisons varier les “**min\_samples**” de 1 à 6.

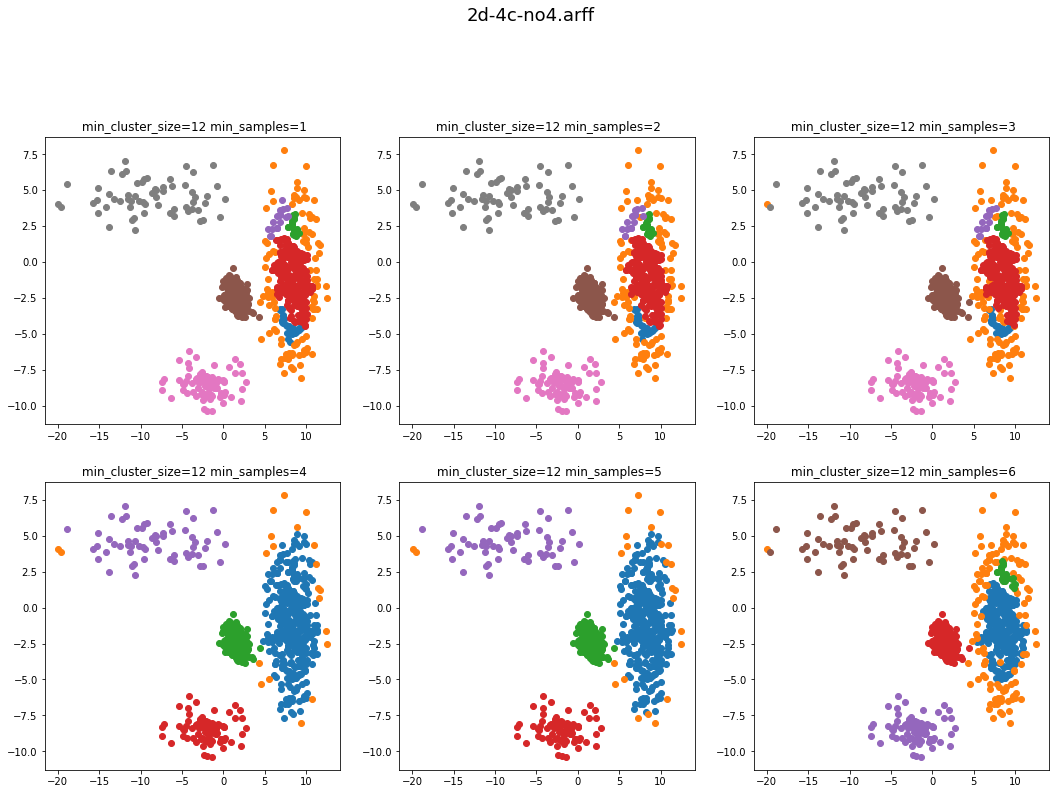


Figure 15

La figure 15 montre qu'en effet, plus les “**min\_samples**” sont élevés, plus les données considérées comme du bruit sont nombreuses, donc les “**min\_samples**” antérieurs à 5 devraient fournir le clustering la plus optimale, mais surtout nous voyons maintenant que la méthode HDBSCAN a réussi à surmonter la limites de la méthode DBSCAN.

À travers cette section, nous pouvons observer que HDBSCAN conserve toujours les avantages qu'il hérite de la version originale DBSCAN, qui est très puissante en travaillant avec les jeux de données avec bruit et fonctionne bien sur les jeux de données de toute forme. En , il peut maintenant corriger les limites de DBSCAN par rapport aux jeux de données à densité inégale en introduisant une approche hiérarchique dans le clustering basé sur la densité.

Malgré ses performances robustes, il est encore difficile de choisir les paramètres optimaux, ce qui nécessite de nombreux essais. C'est pourquoi cette méthode ne doit être utilisée que dans des cas particuliers, comme les jeux de données avec bruit ou à géométrie non plane. En attendant, il serait beaucoup plus simple d'utiliser une méthode générale, tel que le K-Means Clustering, pour les cas généraux.

# Synthèse

Dans cette partie, nous allons essayer les différentes méthodes vues précédemment pour des jeux de données “réels” dont on ne connaît pas le résultat exact :

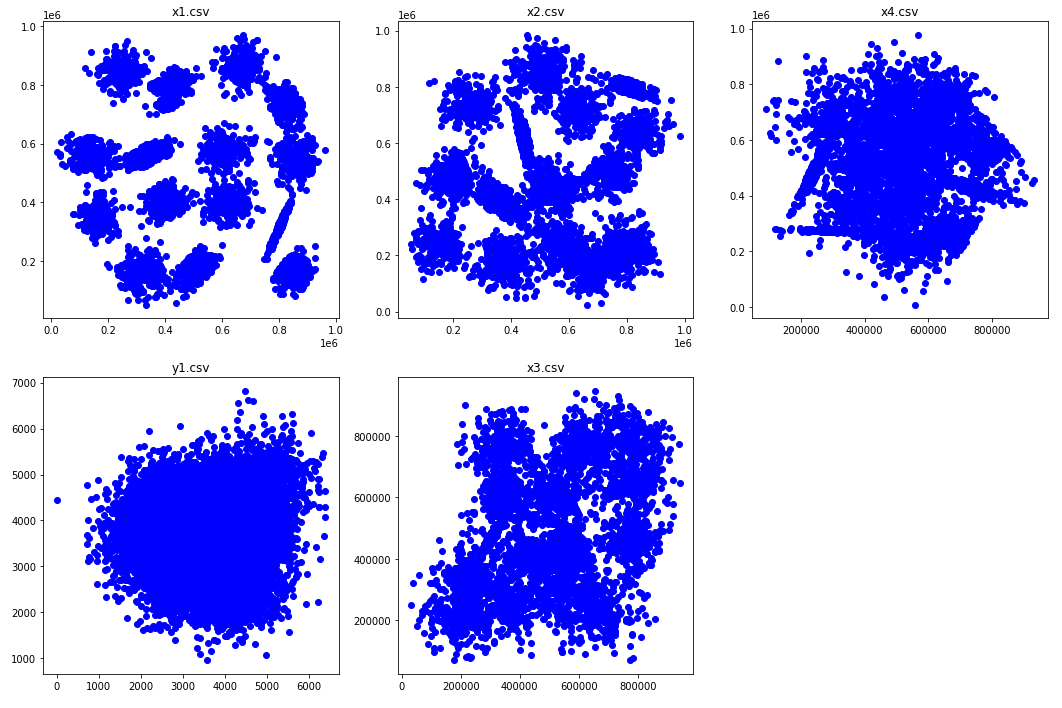


Figure 16

**K-means :**

Tout d’abord, on essaye de trouver le nombre de clusters pour chaque jeu de données en utilisant le score de Davies Bouldin. On itère donc sur sur le nombre de clusters et on choisit le nombre où le score est le plus bas (pour Davies Bouldin) pour chaque jeu de données. On trouve donc que le nombre de cluster est, pour chaque jeu de données :

{'x1.csv': 15, 'x2.csv': 15, 'x4.csv': 14, 'y1.csv': 9, 'x3.csv': 15}

On performe ensuite la méthode de K-means sur les jeux de données en utilisant le nombre de cluster ci-dessus en affichant les résultats :

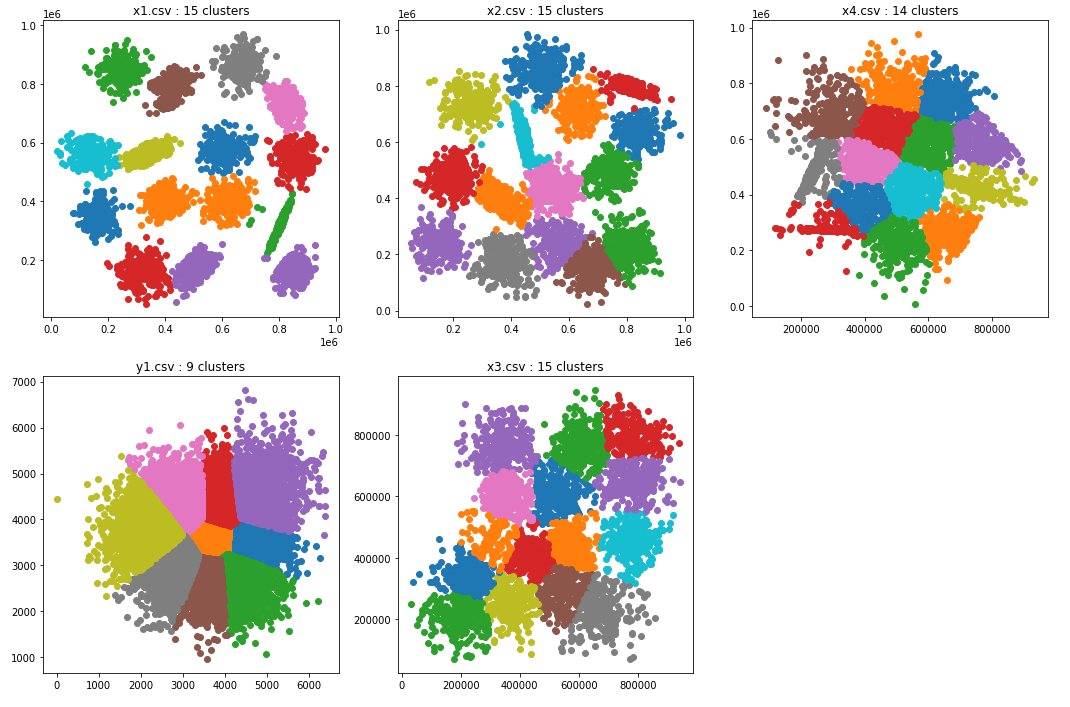


Figure 17

On remarque donc que le résultat est assez satisfaisant pour x1.csv et x2.csv où les clusters sont un peu éloignés les uns des autres et où les points sont bien regroupés, mais les résultats sont moins cohérents sur x3.csv, x4.csv et y1.csv.

**Agglomératif :**

(*nous n’avons pas pu avoir de résultat pour y1.csv pour cette méthode : mémoire RAM insuffisante*)

Comme dans la partie précédente, on essaye de trouver le nombre de clusters et la méthode de linkage avec le score Davies Bouldin. On obtient les résultats suivants :

{'x1.csv': [17, 'average'], 'x2.csv': [16, 'average'], 'x4.csv': [2, 'single'], 'x3.csv': [2, 'single']}

On applique ensuite la méthode de clustering agglomératif sur les jeux de données avec les paramètres trouvés : (figure 18)

On remarque encore une fois que la méthode fonctionne plutôt bien pour x1.csv et x2.csv mais qu’elle fonctionne mal pour x3.csv et x4.csv avec les paramètres proposés.

On va donc réitérer la méthode de clustering agglomératif sur x3.csv et x4.csv mais avec 500 clusters cette fois pour neutraliser le bruit, qui ne mobilisera pas cette fois la totalité des clusters (-1) (car les points qu’on appelle bruit sont plus éloignés du bloc constituant la totalité des clusters, et monopolisent donc les clusters de la méthode). On trouve cette fois : (figure 19)

Même si le résultat n’est pas parfait, on arrive déjà mieux à distinguer des clusters, plus cohérents qu’avec la méthode k-means.

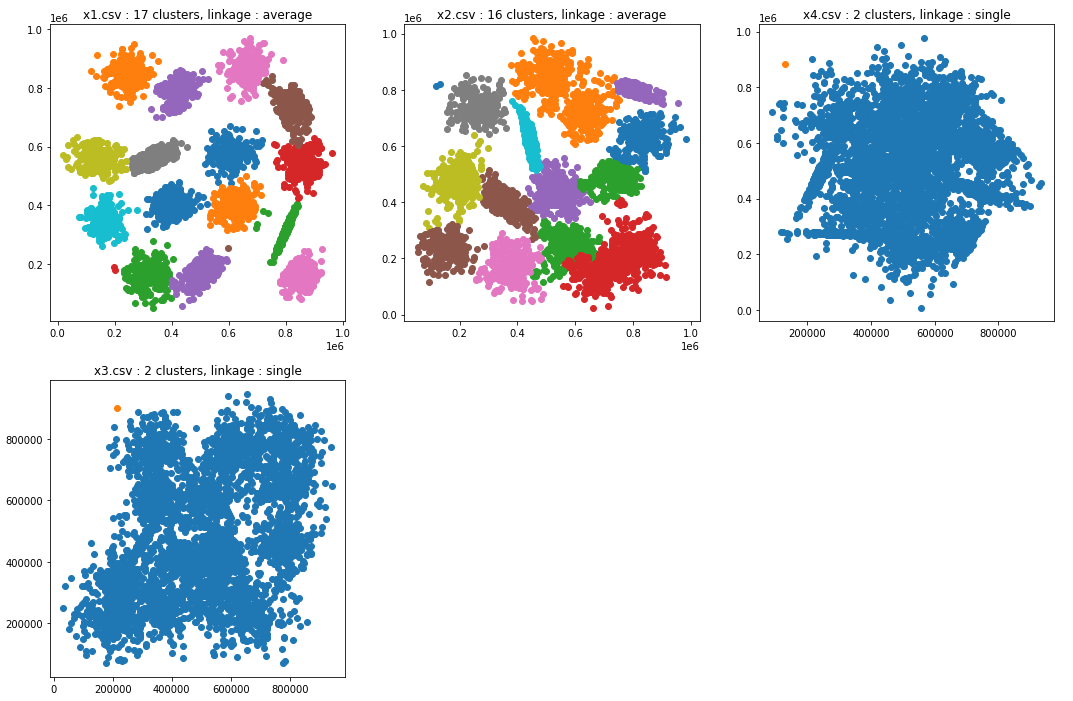


Figure 18

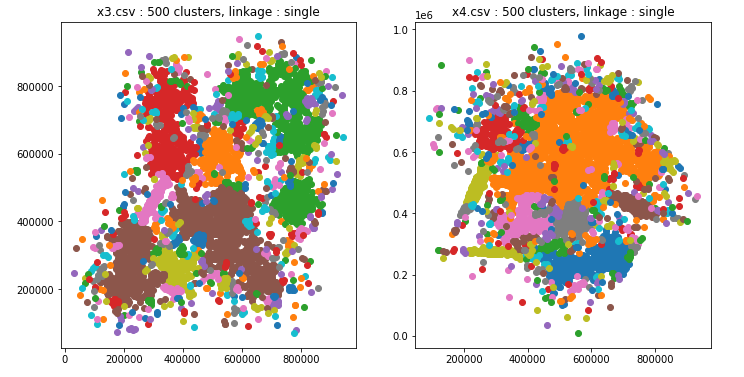


Figure 19

**DBSCAN :**

(*même problème pour y1.csv*)

DBSCAN doit prendre deux paramètres pour fonctionner. eps et min\_samples. Premièrement nous avons essayé de trouver ces paramètres itérativement en utilisant encore une fois le score de Davies Bouldin. Cependant cette méthode est beaucoup trop longue et le score de Davies Bouldin ne permet pas de trouver forcément des paramètres satisfaisants (il aurait été possible d’essayer avec d’autres scores, comme le score silhouette, mais la méthode reste tout de même trop longue, pour des résultats peu satisfaisants).

Nous avons donc dans un deuxième temps utilisé des paramètres choisis par nous-même en fonction du jeu de données :

para["x1.csv"]=[0.03e6,20]

para["x2.csv"]=[0.02e6,9]

para["x3.csv"]=[15000,10]

para["x4.csv"]=[0.02e6,22]

(syntaxe : para[jeu de données]=[eps, min\_samples])

Finalement on trouve :

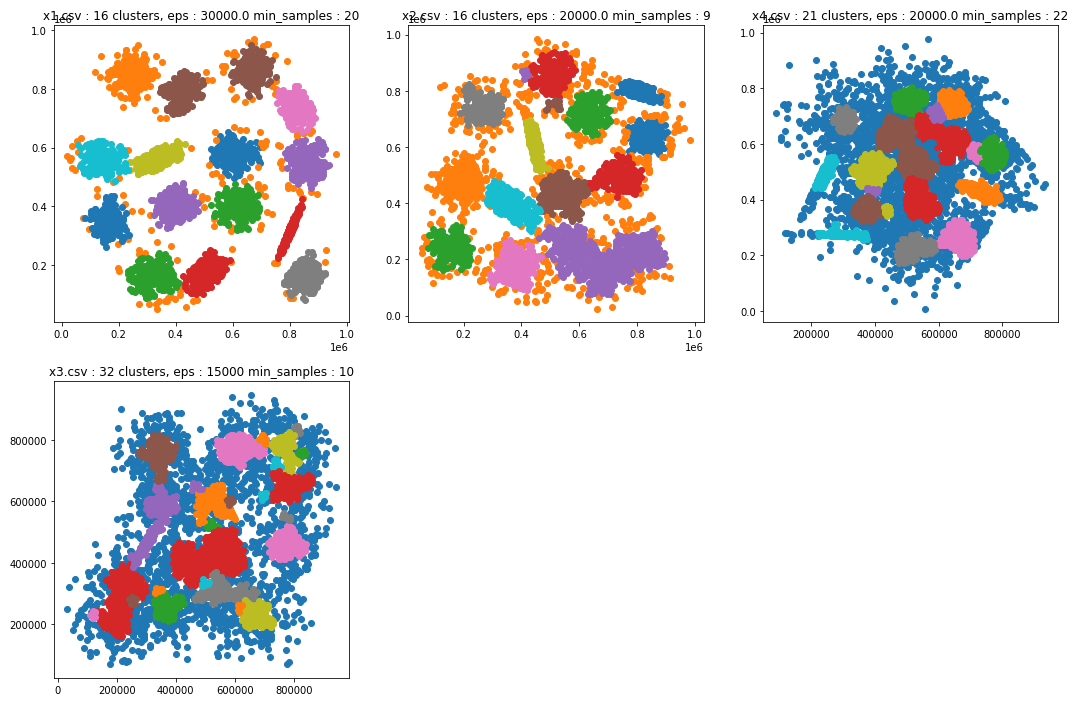


Figure 20

Les résultats sont toujours loin d’être parfait mais on arrive tout de même à distinguer certains clusters pour x3.csv et x4.csv. Pour x1.csv et x2.csv, les résultats sont moins bons qu’avec les méthodes précédentes, mais restent satisfaisants.

**HDBSCAN :**

Pour cette méthode, on choisit nous même le paramètre “**min\_cluster\_size**”. Après application de la méthode HDBSCAN on obtient :

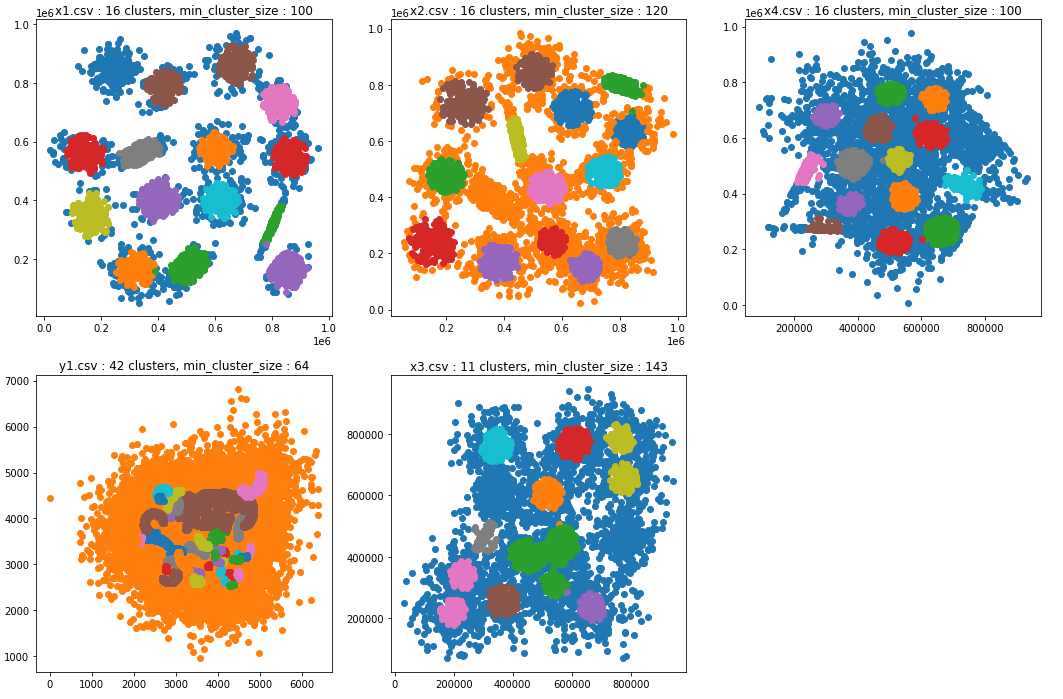


Figure 21

On remarque que la méthode fonctionne toujours assez bien pour x1.csv et x2.scv. Pour x3.csv et x4.csv on trouve des résultats un peu moins satisfaisants qu’avec DBSCAN (clusters plus petits) mais qui concordent. Et pour y1.csv on trouve un résultat beaucoup plus cohérent qu’avec la méthode k-means.

1. [*https://github.com/deric/clustering-benchmark*](https://github.com/deric/clustering-benchmark) [↑](#footnote-ref-0)
2. [*https://github.com/deric/clustering-benchmark*](https://github.com/deric/clustering-benchmark) [↑](#footnote-ref-1)