

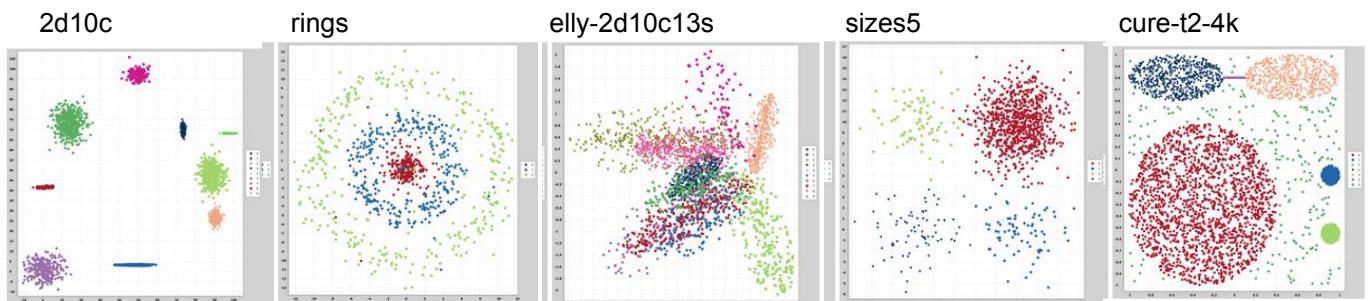
Rapport de TP

Apprentissage non supervisé

1. Jeux de données

Afin de tester les différents algorithmes de clustering et de comparer leur efficacité dans différentes configurations, nous avons sélectionné cinq jeux de données portant des caractéristiques différentes.

Le premier sera notre jeu de données de références, les données sont convexes, bien séparées, la densité est homogène et elles sont non bruitées (2d10c). Le deuxième est un jeu de données non convexe puisqu'il s'agit de 3 cercles imbriqués (rings). Le troisième est composé de données mal séparées, les clusters se superposent (elly-2d10c13s). Le quatrième est constitué de données dont les clusters possèdent des densités variables (sizes5). Enfin, le dernier est un jeu de données bruité (cure-t2-4k).

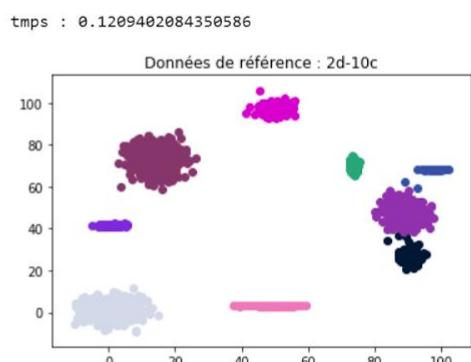


2. Clustering k-means

2.1. Premiers tests sur un jeu de données de référence

Le premier algorithme que nous allons étudier est l'algorithme K-means. Cette méthode détermine k centres dans le jeu de données, et attribue chaque point au centre le plus proche. Ici, nous avons utilisé la méthode d'initialisation des centres 'k-means++'. Après chaque itération, les centres des clusters sont réattribués en fonction des points précédemment ajoutés.

Afin de le tester dans un cas optimal, nous allons utiliser notre jeu de données de référence (2d10c). De plus, nous allons donner en paramètre le nombre exact de cluster attendu : 9.



Nous obtenons ici un très bon résultat puisqu'on peut identifier distinctement les clusters. Par ailleurs, l'exécution est très rapide, avec environ 0,06s de temps d'exécution.

2.2. Estimation du nombre de clusters pour une jeu données de référence à l'aide de critères d'évaluation

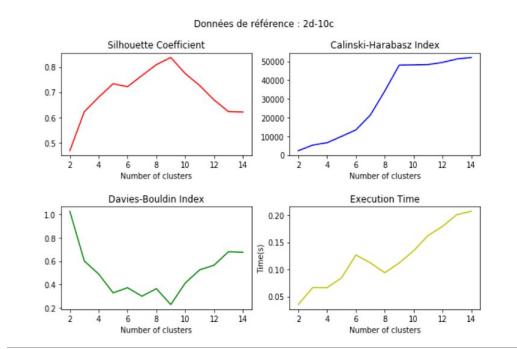
A présent, nous chercherons à déterminer le nombre correct de cluster à l'aide de critères d'évaluation. Nous avons choisi trois critères : le *Silhouette Coefficient*, le *Calinski-Harabasz Index*, et le *Davies-Bouldin Index*. Ces critères ont été retenus dans la liste des critères offerts par scikitlearn puisqu'il ne demandent pas de "vérité" afin de comparer les résultats. Ils sont donc pertinent puisqu'en apprentissage non supervisé, nous n'avons pas de label de référence, on n'effectue pas les prévisions en fonctions de labels connus.

Le coefficient Silhouette prends des valeurs entre -1 et 1. Il est calculé à l'aide de la distance moyenne entre clusters (a) et de la distance moyenne au plus proche cluster(b) selon le calcul suivant : $(b - a) / \max(a, b)$. Il mesure la cohésion des clusters - est ce qu'un point appartient au bon cluster? - et la séparation des clusters - sont-ils bien éloignés ? -. Un coefficient proche de 1 signifie que les cluster sont denses et bien séparés. Si il est proche de 0, alors les clusters se chevauchent. Enfin, un coefficient négatif indique qu'un échantillon a mal été assigné à un cluster.

L' indice de Calinski-Harabasz calcule le ratio entre la dispersion intra-cluster et inter-clusters. Plus cet indice est élevé, plus les clusters sont denses et bien séparés.

Enfin, l'indice de Davies-Bouldin est défini grâce au ratio intra-cluster/inter-clusters des clusters les plus similaires. Il mesure ainsi l'homogénéité et la séparation des clusters. Le score minimum est de 0 et indique que les clusters sont bien espacés et denses.

Nous avons testé ces trois scores sur notre jeu de données de référence (2d10c) afin de retrouver le nombre de cluster correct. Pour cela, nous avons calculé les scores pour des prévisions k-means sur différents nombre de clusters, entre 2 et 15.

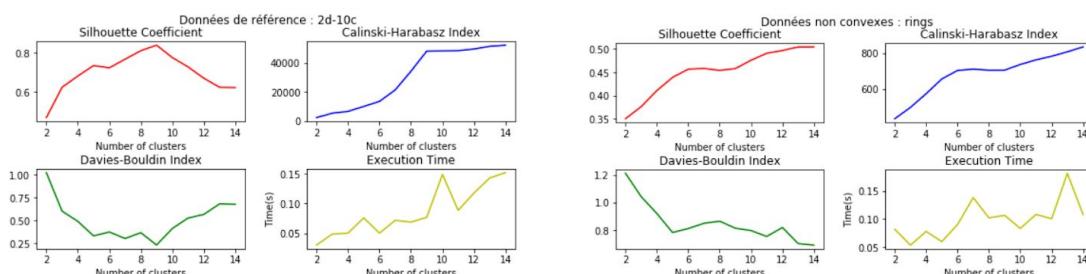


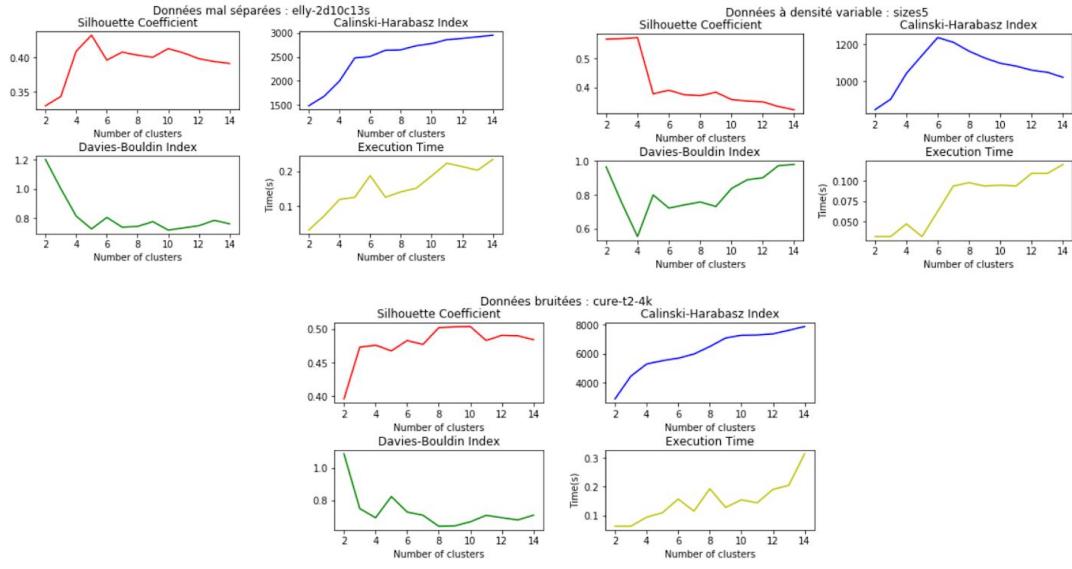
On remarque que, pour les trois scores calculés, on arrive à déterminer le bon nombre de cluster pour ce jeu de données. En effet, pour le coefficient Silhouette, on constate un pic à 0,9 pour 9 clusters. Pour l'indice de Calinski-Harabasz, on remarque également un pic autour de 50000 pour 9 clusters. Cependant, ce score donne de bons résultats également pour plus de 9 clusters, il ne faut donc considérer ses résultats avec précaution. Enfin, l'indice de Davies-Bouldin observe un score le plus proche de 0 pour 9 clusters également, ce qui confirme ce chiffre. En revanche, il donne de bon scores pour un nombre de cluster compris entre 5 et 9. Il faudra donc également considérer ses résultats précautionneusement.

Ainsi, ces scores nous permettent de déterminer un estimation du nombre correcte de clusters pour un jeu de données. Cependant, un score n'étant pas forcément fiable, il faudra considérer les trois afin d'obtenir une estimation plus juste.

2.3. Estimation du nombre de clusters pour différents jeux données à l'aide de critères d'évaluation

Appliquons maintenant cette stratégies aux autres jeux de données, afin d'estimer leur nombre de cluster et de tester la justesses de nos scores. Nous avons calculé les scores des trois critères d'évaluation en fonction du nombre de clusters considérés sur tous nos jeux de données.





Et voici les nombres de clusters attendus :

Jeu de données	Référence	Non convexe	Mal séparées	Densité variable	Bruitées
Nombre de clusters	9	3	10	4	7

Pour les données de référence, nous avons déjà vu dans la partie précédente que l'on identifiait facilement le nombre correcte de clusters. Mais qu'en est-il des autres ?

Le seul jeu de données pour lequel on arrive à déterminer le nombre de clusters est celui ayant des densités variables. On remarque, pour le coefficient de Silhouette, un pic jusqu'à 4 clusters et qui chute par la suite. Pour l'indice de Calinski-Harabasz, lui, donne de meilleurs résultats pour 6 clusters, mais en observant l'écart de score entre 4 et 6 clusters, on constate qu'il est inférieur à 200, ce qui n'est pas énorme compte tenu de l'échelle. Enfin, pour l'indice de Davies-Bouldin, on situe clairement un pic le plus proche de 0 pour 4 clusters. En croisant les résultats de ces trois critères d'évaluation, on peut déterminer un nombre de cluster égal à 4 pour ce jeu de données. Cette supposition est confirmée par nos attentes.

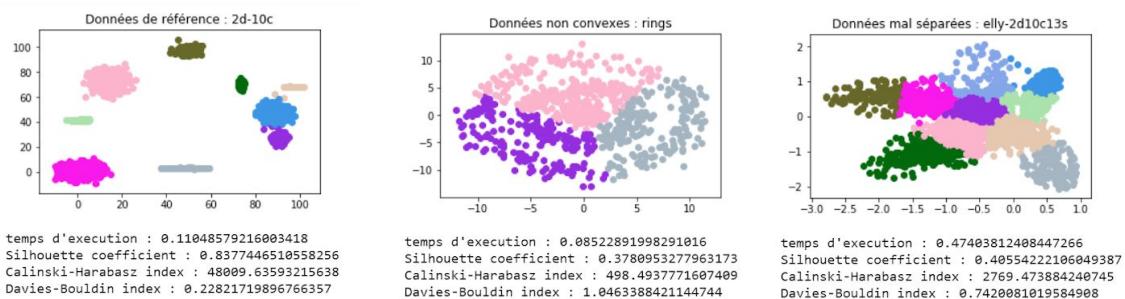
Cependant, tous les autres jeux de données sont difficiles à analyser avec l'algorithme k-means. En effet, les différents indices étudiés ne démontrent pas de pics net, ou ne se vérifient pas entre eux.

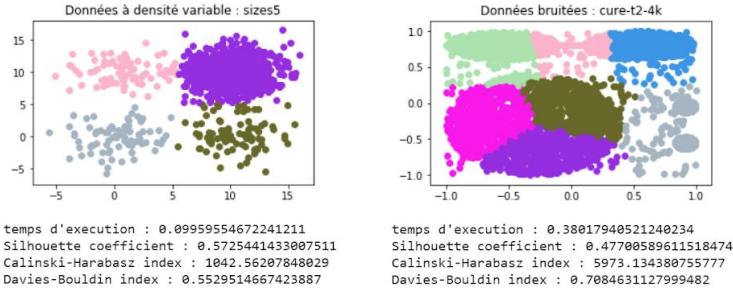
Ces résultats semblent montrer que l'algorithme k-means ne parvient pas à prédire de bons clusters pour des jeux de données non convexes, mal séparés ou bruités.

2.4. Application de l'algorithme k-means sur différents jeux de données

Après avoir estimé le nombre de clusters pour de chaque jeu de données avec l'algorithme k-means, nous avons pu constater que celui-ci était difficile à trouver pour certains jeux de données non convexes, mal séparés ou bruités.

Nous allons à présent appliquer l'algorithme k-means sur nos jeux de données, en lui renseignant le nombre de clusters attendu. Regardons si cet algorithme, connaissant le nombre de cluster à déterminer, va réussir à catégoriser correctement les données.





Comme vu précédemment, le jeu de données de référence sont très bien classifiés avec l'algorithme k-means. Pour le dataset à densité variables (sizes5), les clusters sont également bien déterminés, la densité des données n'influe donc pas sur la performance de cet algorithme.

En revanche, pour des données non convexes avec des cercles imbriqués, k-means n'arrive pas à déterminer correctement les clusters, la convexité semble donc être un critère important pour cet algorithme. De même, des données mal séparées donne de mauvais résultats : visuellement on remarque que k-means ne fais pas se chevaucher les clusters. Enfin, le dernier jeu de données comportant du bruit, révèle également de mauvaises performances de la part de l'algorithme k-means. Même en ayant le bon nombre de clusters, cet algorithme n'arrive pas à déterminer les clusters correctement.

En effet, l'algorithme k-means attribuant les clusters de chaque point en fonction du plus proche centre, n'arrivera pas à attribuer des "centres" pour des données non convexes. Par exemple, avec le jeu *rings*, les données formant des cercles imbriqués, on remarque que l'algorithme a découpé les données en clusters convexes.

Par ailleurs, le test sur le jeu de données ayant des clusters se chevauchant montre également que k-means n'est pas adapté à ce type de données. De plus, les clusters étant de forme ovale/allongée, l'algorithme aura plus de difficultés à les déterminer correctement. Par exemple, un point étant à une extrémité d'un cluster A allongé pourra être attribué à un cluster B si son centre est plus proche.

On retrouve ce schéma pour les données *cure-t2-4k* où les deux clusters ovales ont été mal déterminés (ici en vert/rose/bleu).

Enfin, on peut également se demander quelle est le rôle de l'initialisation dans la performance de l'algorithme. En effet, si on prend ici aussi en exemple le jeu *cure-t2-4k*, on constate que le plus gros cluster circulaire a été divisé en trois clusters (fushia, violet et kaki). On peut supposer que trois des centres ont été initialisés au sein de celui-ci. Si un seul centre avait été déterminé au centre du cercle, le cluster aurait peut-être pu être correctement prédit.

Ainsi, après avoir étudié et analysé les résultats des tests de l'algorithme k-means, on peut en déduire que cette méthode est très efficace pour des données convexes, non bruitées et ayant des formes plutôt circulaires ou ramassées. La densité des données, en revanche, n'est pas un critère et k-means démontre de bons résultats qu'elle soit homogène ou non.

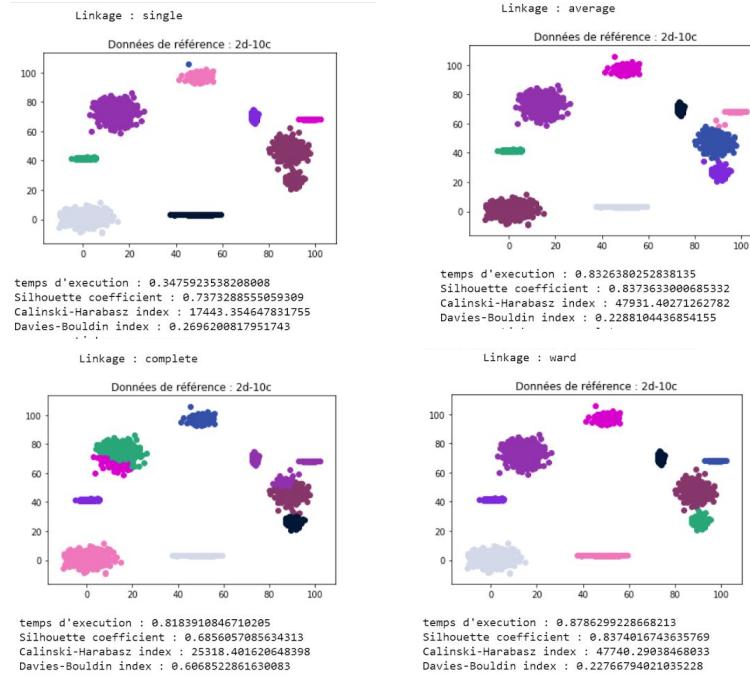
3. Clustering agglomératif

3.1. Application de la méthode agglomérative et test des combinaisons possible en connaissant le nombre de cluster

Dans cette partie, nous allons étudier une méthode de clustering dite "agglomérative" ou de clustering ascendant. Cette méthode part du principe que chaque point appartient à un cluster différent. L'algorithme va ensuite fusionner les plus proches afin d'obtenir des clusters plus grands.

D'autre part, cette méthode peut combiner les clusters de manière différente, c'est à dire avec des critères de distance différents. La première méthode, *single*, utilise la distance minimum entre clusters. La méthode *average* utilise la distance moyenne. *Complete* est une méthode qui utilise la distance maximale, et enfin *ward* minimise la variance des clusters. Dans notre étude, nous considérerons uniquement la distance Euclidienne.

Tout d'abord, nous la testerons sur notre jeu de données de référence connexe afin de l'étudier dans des conditions optimales.

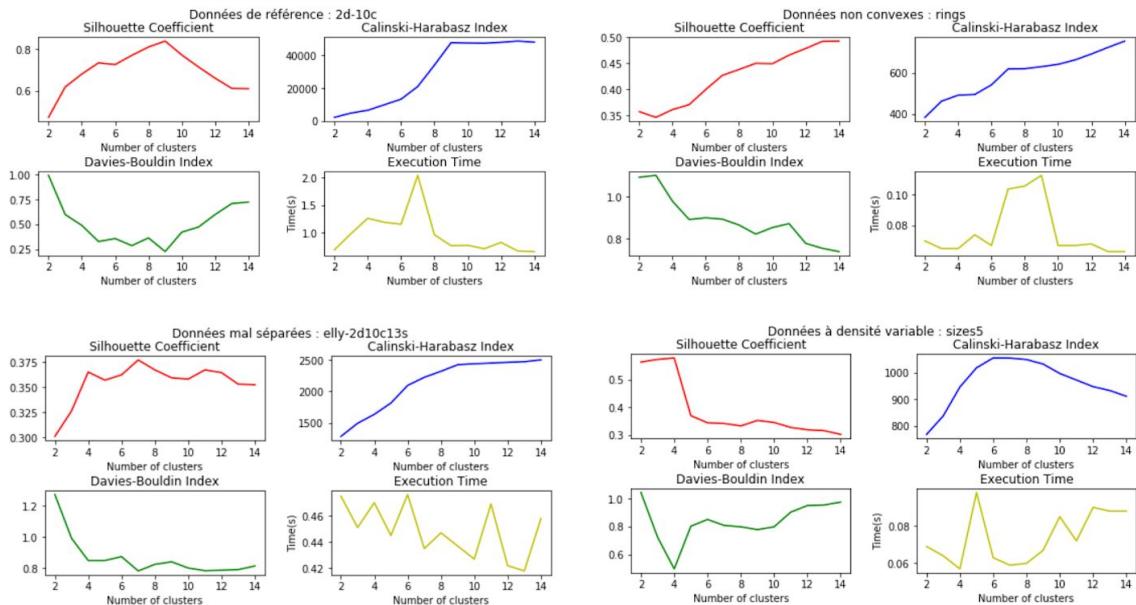


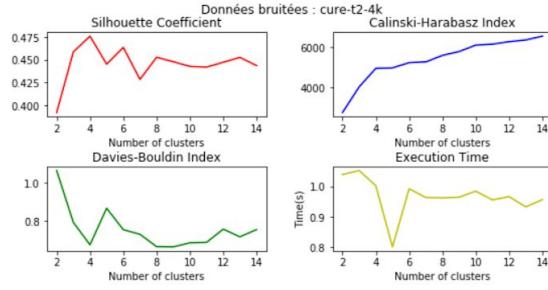
Après avoir effectué les tests, on constate que, pour ce jeu de données, les linkages *single* et *complete* offrent de mauvaises performances. En effet, on le remarque visuellement, avec le cluster situé en bas à droite qui a été mal déterminé. De plus, avec les critères d'évaluation Silhouette, Calinski-Harabasz et Davies-Bouldin décrits précédemment, on remarque clairement que leur score est moins bon. En revanche, les méthodes *ward* et *average* démontrent de meilleurs résultats. Les clusters semblent bien déterminés visuellement, et les scores sont presques équivalents : autour de 0,83 pour le coefficient Silhouette, autour de 47000 pour l'indice de Calinski-Harabasz, et autour de 0,22 pour l'indice de Davies-Bouldin.

Pour cette raison, et afin de limiter notre nombre de tests par la suite, on décide de conserver la méthode de combinaison des clusters *ward*. On aurait également pu choisir *average*.

3.2. Estimation du nombre de clusters de nos jeux de données à l'aide de la méthode agglomérative

A présent, nous allons, comme avec l'algorithme k-means, tenter de déterminer le nombre de clusters de chaque jeu de données. Pour cela, nous allons encore une fois nous appuyer sur les critères d'évaluation de Silhouette, Calinski-Harabasz et Davies-Bouldin. Nous allons itérer la méthode d'agglomération afin de chercher le nombre de clusters obtenant les meilleurs scores.





Rappel des clusters attendus

:

Jeu de données	Référence	Non convexe	Mal séparées	Densité variable	Bruitées
Nombre de clusters	9	3	10	4	7

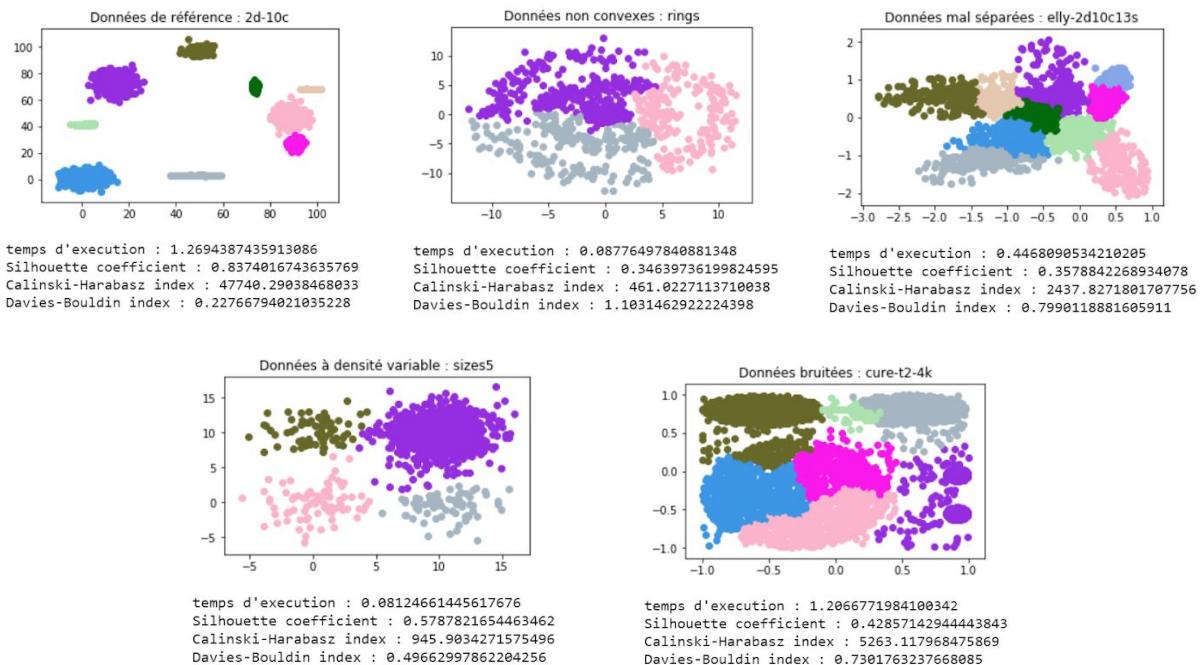
Ici encore, on retrouve des scores nous permettant d'identifier le nombre correct de clusters pour notre jeu de données de référence et pour celui ayant des densités variables. En revanche, pour des clusters non convexes, mal séparés ou bruités, cet algorithme ne semble pas pouvoir déterminer de bons clusters, au vu des critères que nous avons choisi.

Nous allons donc à présent appliquer la méthode agglomérative à tous nos jeux de données afin d'observer la qualité de ses clusters en fonction du type de données.

3.3. Application de la méthode agglomérative à tous nos jeux de données

Enfin, après avoir tenté d'estimer le nombre de clusters pour chaque jeu de données, nous avons encore une fois constaté que la méthode agglomérative de parvenait pas à de bons résultats pour les jeux de données non convexes, bruité ou mal séparés.

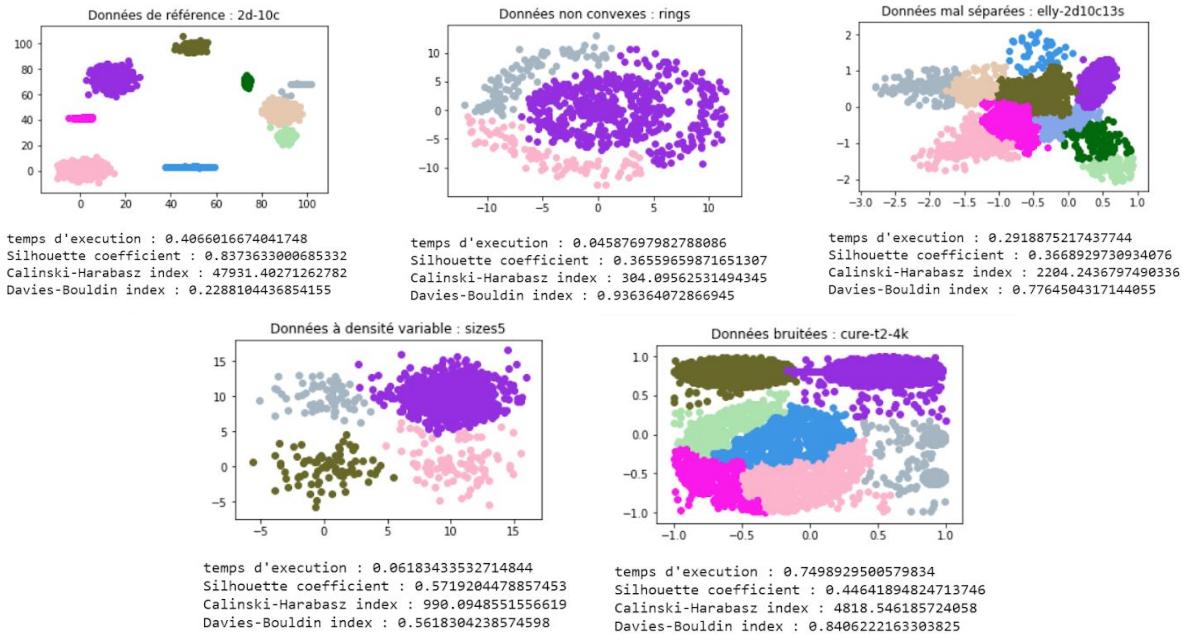
Nous allons à présent appliquer cette méthode en lui renseignant le nombre de clusters, afin d'observer si les clusters sont correctement déterminés. Nous choisissons ici aussi de travailler avec le linkage *ward*.



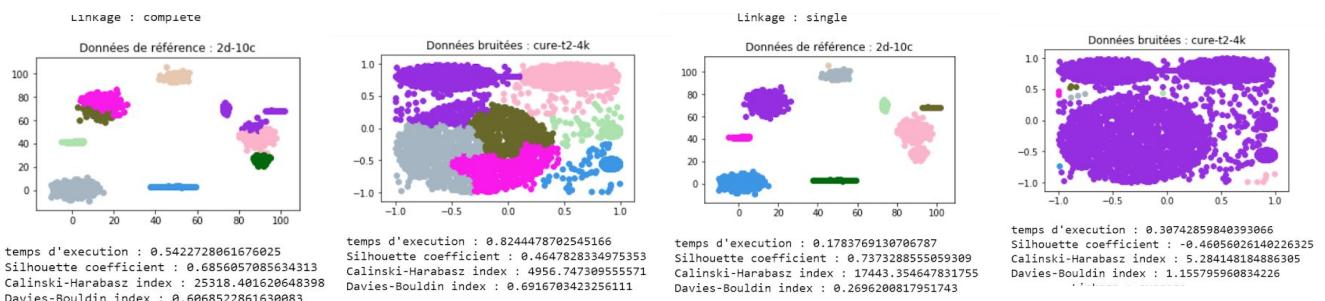
On constate ici que, encore une fois, les données de référence et les données à densité variables sont les seuls à obtenir de bons résultats. En effet, les données non convexes, mal séparés, et bruités ne révèlent pas de clusters satisfaisant avec cette méthode agglomérative, et ce, même avec le bon nombre de clusters renseigné.

Par ailleurs, nous avons effectué les mêmes tests avec les autres méthodes de linkage, *average*, *complete*, et *single*.

On remarque que la méthode *average* montre de bons résultats, similaires à ceux de *ward* :



Nous ne présenterons pas ici tous les résultats des deux autres fonctions puisqu'elles donnaient des résultats trop éloignés de ce que l'on recherchait, même pour le jeu de données de référence:



Comme précédemment, les méthodes *single* et *complete* donnent de très mauvais résultats, les clusters ne sont pas du tout identifiés. Ce sont des méthodes qui, pour *single*, minimisent la distance entre les clusters les plus proches, et pour *complete*, minimise la distance maximale entre deux clusters. Ces deux techniques ne semblent pas les plus efficaces et on préférera, dans les cas que nous avons étudié, les deux autres méthodes.

D'autre part, on remarque que l'algorithme agglomératif semble moins sensible à la forme des clusters. En effet, ici, les clusters de forme ovale ou allongés sont détectés. On le constate avec le jeu de données *cure-t2-4k*, qui possède deux formes ovales en haut des graphiques, et qui ont été correctement déterminés dans trois méthodes sur les 4 testées. De plus, une hétérogénéité des densités des clusters ne semblent pas poser de problème pour cet algorithme.

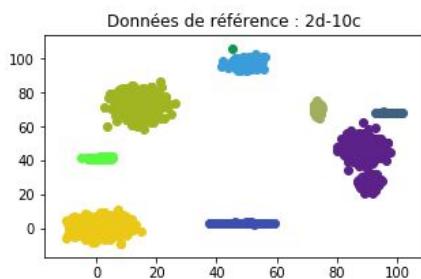
4. Clustering DBSCAN

4.1. Objectifs de DBSCAN

L'intérêt principal de DBSCAN est d'identifier les formes non convexes. Pour cela, il considère pour tous les points combien de points sont à une distance *epsilon* ou moins. Si un nombre de points supérieur à son paramètre *min_sample* respecte ce critère, alors le point est un 'core' du cluster, et tous les points à sa portée font partie du même cluster que lui. Les clusters se propagent donc de voisin en voisin.

4.2. Premiers essais sur le jeu de données de référence

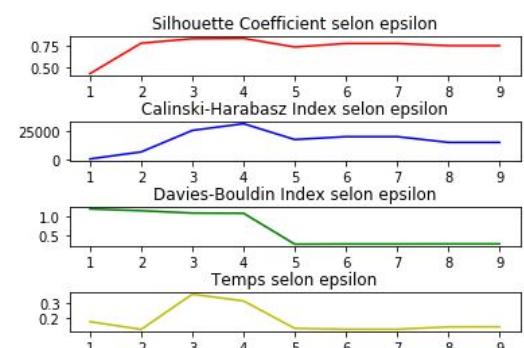
Nous allons tout d'abord essayer d'appliquer cette méthode à notre échantillon de référence aux formes convexe bien identifiées, avec des paramètres choisis aléatoirement (ici, *eps*=5.0 et *min_sample*=10):



Sur l'échantillon de référence avec ces paramètres, les données sont bien clusterisées. On observe tout de même que le cluster supérieur (en bleu) est séparé d'un autre cluster 'bruit' tandis que le cluster violet de droite pourrait potentiellement être séparé en 2. Nous allons voir s'il est possible de changer cela au travers de différents tests sur les paramètres.

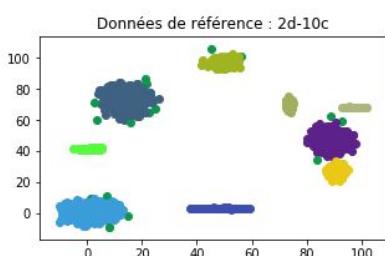
Nous allons maintenant afficher les différents indicateurs (*Silhouette Coefficient*, le *Calinski-Harabasz Index*, et le *Davies-Bouldin Index*, ainsi que le temps) en fonction des valeurs d'*epsilon* que nous ferons varier entre 1 et 10, tout en conservant une valeur de *min_sample* de 10. *Epsilon* est une distance maximale pour laquelle 2 points peuvent être considérés comme voisins. On peut donc penser que diminuer la valeur mènera à une augmentation du nombre de clusters et de la granularité de la classification.

Nous observons que le *Silhouette Coefficient* et le *Calinski-Harabasz index* sont au plus haut pour des valeurs d'*epsilon* aux alentours de 4, et observe que le *Davies-Bouldin index* diminue entre les valeurs d'*epsilon* entre 4 et 5, ce qui indique que nous devrions trouver une valeur exploitable aux alentours de 4.

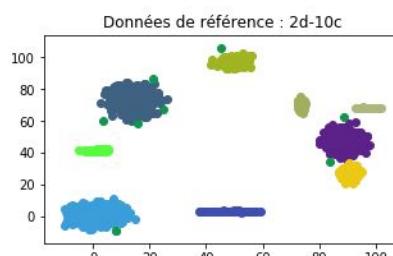


Nous allons afficher les clusters afin de vérifier ces indications:

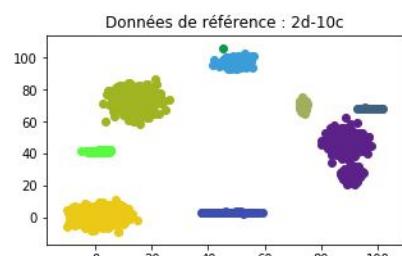
eps = 3.0



eps = 4.0



eps = 5.0



Nous observons en effet que pour $\text{eps} = 4$ les principaux clusters sont séparés comme on le souhaite, tandis que pour $\text{eps}=5$ le cluster violet de droite n'est pas séparé en 2 comme nous le souhaiterions. En revanche, pour $\text{eps}=4.0$ le cluster violet de droite comprend également plusieurs éléments considérés comme du bruit dans ses alentours, une valeur un peu plus élevée permettrait peut-être de remédier à cela, donc la valeur que nous souhaiterions est donc bien comprise entre 4 et 5, comme vu précédemment. Après avoir effectué des tests, 4.4 semble être la valeur la plus élevée pour laquelle nous obtenons une séparation du cluster de droite et qui semble donc être la plus adaptée, même si l'affichage est quasi-similaire à celui pour $\text{eps}=4.0$. Nous utiliserons donc une valeur de $\text{eps}=4.4$ pour la suite.

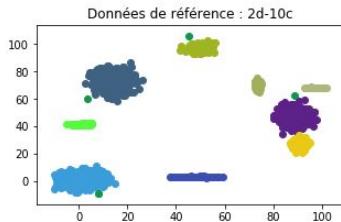
Nous allons maintenant faire varier min_sample entre 1 et 15. Une valeur plus haute devrait entraîner une plus haute granularité en augmentant les prérequis pour qu'un point soit core.

Il est ici simple de remarquer qu'une valeur se démarque: pour un min_sample de 10 minimum le *Silhouette Coefficient* et *Calinski-Harabasz Index* augmentent soudainement. Lorsque l'on affiche les clusters, en dessous de 10 le cluster de droite n'est encore une fois un seul cluster, tandis qu'après 10 il se sépare ce qui est ce que l'on a décidé d'obtenir.

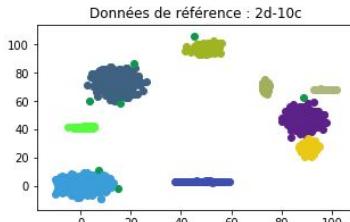
Par malchance, il se trouve que 10 était la valeur que nous avions choisi au départ, donc cette analyse n'a pas changé grand chose à nos valeurs ni notre affichage.

Pour faciliter ce processus de recherche dans le futur et effectuer des vérifications, nous allons évaluer les différents index sur un dataset donné et renvoyer les valeurs qui donnent les meilleurs résultats pour le *Silhouette Score*, le *Calinksyi-Harabasz Index* et le *Davies-Bouldin index* ainsi que tracer les clusters correspondants. Pour cela la fonction utilisée effectuera les prédictions en parcourant les listes de eps et min_sample fournies en argument, afin de faire une recherche plus poussée qu'en variant un seul paramètre à la fois comme jusque-là. Vérifions nos résultats pour eps appartenant à l'intervalle [0.5,30] avec un pas de 0.5 et min_sample appartenant à [1, 80] avec un pas de 3.

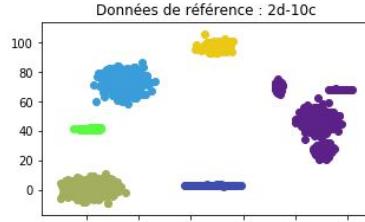
best values according to silhouette:
(4.5, 13) with an index of 0.8357385963035474



best values according to Calinski-Harabasz:
(5.5, 58) with an index of 34111.87890315967



best values according to Davies-Bouldin
(13.0, 1) with an index of 0.25107875781726074



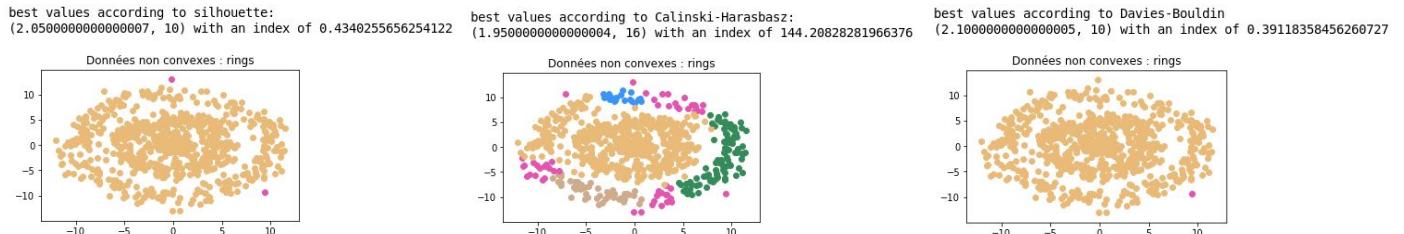
On observe que le *Silhouette Coefficient* est à son plus haut pour $\text{epsilon} = 4.5$ et $\text{min_sample} = 13$, ce qui correspond environ à ce que nous avions trouvé par nous-même. Le graphe associé correspond à ce que l'on recherche. La solution trouvée par le *Calinksyi-Harabasz index* est similaire, avec un peu plus d'éléments identifiés comme du bruit, en raison de son min_sample plus élevé. Enfin le *Davies-Bouldin index* trouve une classification plus large, avec une valeur très élevée de 'epsilon' et très faible de min_sample , ce qui n'est pas ce que l'on recherche. De plus, avec de telles valeurs sur un dataset contenant des groupes de données moins espacés, nous n'obtiendrons qu'un seul cluster. Nous avons donc trouvé des valeurs acceptables, même si imparfaites, à l'aide du *Silhouette Score*.

Une autre manière de procéder pour obtenir des valeurs des paramètres est celle évoquée dans le cours: afficher les distances des k plus proches voisins de chaque point, les trier puis afficher, et sélectionner un epsilon égal à la valeur de la distance correspondant à l'inflexion de la courbe.. Cette méthode est implémentée au travers de la fonction 'find_eps'. Nous utilisons un graphe interactif pour lire les valeurs plus facilement. Une fois la valeur de eps lue, il suffit de parcourir des valeurs de 'min_range' afin d'en trouver une qui convient. Cependant, la valeur à l'inflexion de la courbe est souvent un peu basse. Nous avons vu qu'il fallait essayer de sélectionner une valeur telle que seuls les points que nous estimons être du bruit n'aient pas de voisin.

4.3. Extension à de nouveaux jeux de données

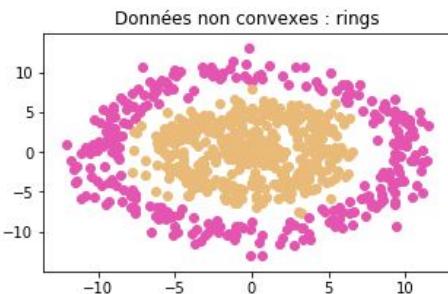
4.3.1 Données non convexes

Dans cette partie, nous allons tester cette méthode sur notre jeu de données non convexes, en commençant par exécuter notre fonction de recherche des meilleures solutions. Après quelques tests préliminaires avec des intervalles larges, nous testons les intervalles ‘eps’ (1.5,2.4,0.05), et ‘min_sample’ (10,60,2). Voici les résultats:



Nous pouvons noter que les meilleures valeurs de *Silhouette* et *Davies-Bouldin* ne donnent pas de bons résultats, les indices ne sont pas très intéressants pour ce dataset. *Calinski-Harabasz* obtient un meilleur clustering, sans pour autant être ce que l'on recherche. Peut-être cherchons-nous leurs maximum/minimum sur un ensemble trop grand et manquons des extrema locaux qui conviendraient mieux.

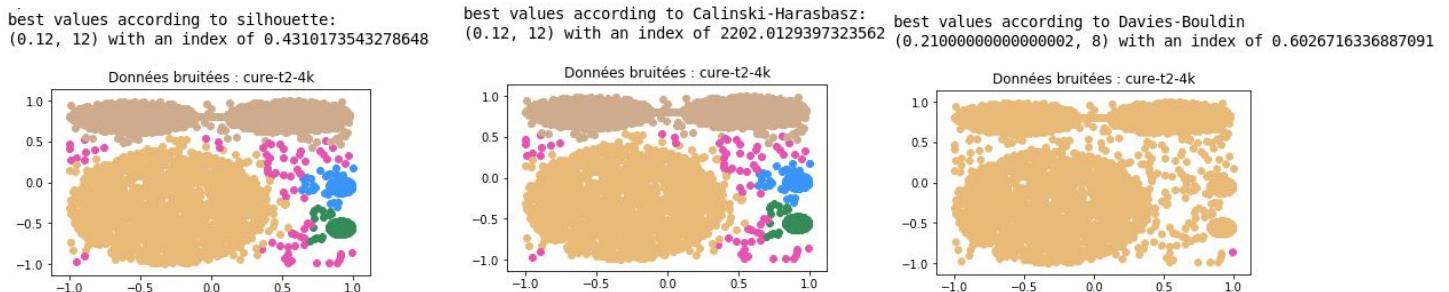
En cherchant manuellement (et à l'aide des fonctions d'affichage des différents index) à partir des valeurs de paramètres données par *Calinski-Harabasz*, nous trouvons un set de paramètres (eps = 2.25, min_sample = 33) pour lesquelles nous parvenons à séparer le cercle extérieur:



Malheureusement, il nous est difficile de séparer les 2 cercles intérieurs qui sont quasiment confondus. L'algorithme opérant sur des voisins, il arrive toujours à passer d'un cercle à l'autre par saut. Des valeurs permettant cette séparation existent cependant peut-être, mais cela semble très complexe car ils se superposent en partie. En d'autres termes, nous avons réussi à reconnaître des formes non convexes, à condition qu'elles soient suffisamment séparées.

4.3.2 Données bruitées

De même, dans cette partie, nous appliquons notre méthode à nos données bruitées. C'est à dire que nous utilisons notre fonction de parcours des valeurs de ‘eps’ et ‘min_sample’, avant de poursuivre une recherche manuelle dans les environs des résultats obtenus. Après plusieurs parcours:



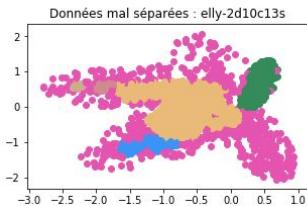
Silhouette et *Calinski-Harabasz* nous donnent un couple de valeurs exploitables où les clusters semblent plutôt bien séparés. Après avoir recherché autour de ces valeurs, nous ne parvenons pas à séparer la masse de points du haut de notre graphe. Cela s'explique par la forte densité du pont qui les relie, qui empêche la distinction avec DBSCAN - qui se base sur les densités-, et passera donc de point en point d'un groupe à l'autre. DBSCAN réalise tout de

même un clustering intéressant, puisqu'il est facile d'ignorer le bruit avec DBSCAN car ses paramètres permettent de définir une densité en dessous de laquelle des points vont être considérés comme du bruit.

4.3.3 Données mal séparées

Nous réutilisons encore une fois notre méthode, et seul *Calinski-Harabasz* nous donne un résultat exploitable:

best values according to Calinski-Harabasz:
(0.1, 16) with an index of 305.73910682256917

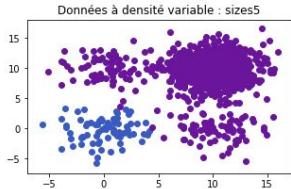


Ce clustering est issu d'une valeur d'epsilon basse et une valeur de min_sample relativement haute. Cela permet de détecter les différentes zones ayant une certaine densité, séparées par des zones de densité plus faible considérées comme du bruit. Normalement, DBSCAN n'est pas très efficace dans ce cas, mais avec des paramètres très particuliers, cette méthode permet d'extraire des clusters de plus haute densité.

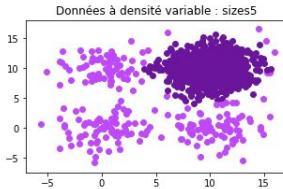
4.3.4 Données à densité variable

Après avoir appliqué notre fonction de recherche de paramètres, puis effectué de la recherche manuelle, nous obtenons ce type de résultats:

best values according to silhouette:
(2.9, 18) with an index of 0.5701041745532874



best values according to Calinski-Harabasz:
(1.55, 24) with an index of 695.6537946314494



Dans le cas de gauche, la densité paramétrée dans notre DBSCAN est suffisamment basse pour identifier le cluster d'en bas à gauche comme n'étant pas du bruit, mais cela a pour effet de mettre le reste des points dans le même cluster. En effet, ils sont reliés par des zones de densité similaire. Dans le cas de droite, c'est l'opposé : la densité est élevée et ne reconnaît que le cluster violet, considérant les autres points comme du bruit. Cela est particulier car les groupes sont mal séparés, mais illustre la difficulté de DBSCAN: il est compliqué à utiliser si les clusters se différencient par leur densité tout en étant mal séparés.

4.4 Conclusion DBSCAN

DBSCAN semble donc être tout indiqué quand nos clusters à identifier sont séparés, que ce soit par du vide ou des zones à densité plus faible que la densité de notre cluster le moins dense. En revanche, comme il se construit par sauts de point en point, la forme des clusters n'importe absolument pas, tant que notre cluster va conserver une densité supérieure à celle du bruit qui l'entoure, et cela même pour les clusters non convexes. Notre méthode, même si plutôt 'bruteforce' semble tout de même être efficace car il nous est difficile de trouver de meilleurs clusterings manuellement ou à l'aide de la méthode de traçage de la courbe de distance du plus proche voisin. Cette méthode est cependant difficilement applicable sur des données de plus grande taille, car les temps de calcul en double parcours pourraient devenir bien plus élevés. Dans ce cas, il sera judicieux d'utiliser la méthode de traçage des distances des plus proches voisins pour déduire epsilon, puis faire varier min_samples uniquement.

5. Clustering HDBSCAN

5.1 Objectifs de HDBSCAN

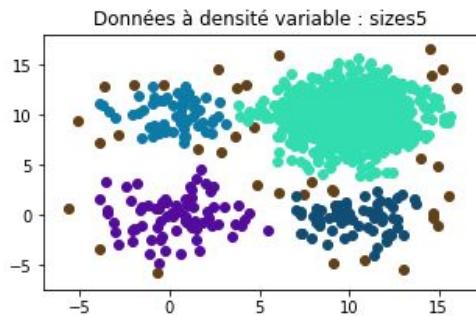
HDBSCAN a pour objectif d'être efficace face au bruit et aux variations de densité. En effet, nous avons vu que DBSCAN, triant les éléments par simple lien, pouvait avoir des difficultés à séparer 2 groupes si un élément de bruit agissait comme 'pont' entre les clusters par exemple. Ce défaut va être corrigé par HDBSCAN en changeant la fonction établissant la distance entre différents points, pour qu'elle soit fortement défavorable aux points de bruit.

Alors que DBSCAN va considérer tout point de densité plus faible que la ‘cut value’ passée en paramètre comme bruit globalement, HDBSCAN va (en pratique) considérer des points comme bruit selon un critère variable dans l'espace.

5.2 Données à densité variable

Nous adaptons notre algorithme de parcours des paramètres pour HDBSCAN, puis l'utilisons sur le set de données à groupes de densité variables. Voici le résultat obtenu (résultat similaire pour *Silhouette* et *Calinski-Harabasz*) :

best values according to silhouette:
 $(2, 10)$ with an index of 0.5522732426255245



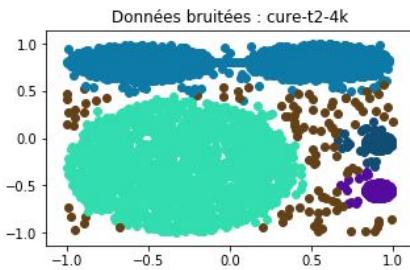
Les clusters obtenus pour `min_sample=2` et `min_cluster_size` nous conviennent, les éléments traités comme bruit sont clairement séparés des clusters et chaque groupe est identifié comme un cluster différent.

HDBSCAN obtient de très bon résultats sur ce set de données pourtant très complexe, avec une densité variable et une séparation spatiale peu évidente, ainsi que des clusters reliés par des “ponts de bruit”. Rappelons nous que nous ne pouvions pas obtenir ces résultats à cause de l’unique ‘cut value’ de densité de DBSCAN. HDBSCAN semble donc être bien plus approprié pour le clustering à densité variable.

5.3 Données bruitées

Nos recherches nous donnent le type de résultat suivant:

best values according to silhouette:
 $(45, 2)$ with an index of 0.42840056451711483



Les clusters sont bien identifiés, à quelques imprécisions près. En effet, les clusters comprennent une petite quantité de ‘bruit’, mais nous n'avons pas trouvé de valeurs permettant de diminuer cet effet sans briser la clusterisation des groupes supérieurs.

Le ‘pont’ reliant les groupes supérieurs est un problème car pour la plupart des paramètres, sa haute densité fait que les groupes adjacents sont considérés en grande partie comme du bruit. Cela contraint fortement les valeurs que peuvent prendre nos paramètres, ce qui est regrettable car avec des paramètres différents les autres clusters sont bien plus facilement identifiables et comprennent moins de bruit.

Nos expériences nous font supposer que ce set de données serait en réalité très facile à traiter en l'absence de ce “pont” de haute densité, et que HDBSCAN est en effet résistant au bruit.

5.3 Conclusion HDBSCAN

HDBSCAN a un comportement à première vue similaire à DBSCAN, groupant les éléments selon la densité. Cela a pour conséquence qu'il va avoir des propriétés similaires, comme le fait d'être insensible à la forme que doivent prendre les différents clusters. Seulement, il va être bien plus efficace lorsque nos différents clusters vont varier en densité, tout en améliorant la résistance au bruit de DBSCAN. L'algorithme est cependant plus complexe et coûteux en temps que le simple lien effectué par DBSCAN. Une conséquence de cette complexité est également la difficulté à comprendre l'impact que peuvent parfois avoir de petites variations des paramètres, contrairement au cas de DBSCAN où il est très facile d'imaginer les conséquences d'un changement.

6. Comparaison des méthodes de clustering

Ainsi, après l'étude de quatres méthodes de clustering différentes : k-means, agglomérative, DBSCAN et HDBSCAN, nous pouvons conclure sur leur efficacité et leur adaptabilités à différents jeux de données. Afin de juger de leur qualité de clustering, nous avons utilisé trois scores : Silhouette, Calinski-Harabasz et Davies-Bouldin. Cependant, il faudra garder à l'esprit que ces critères ne sont pas fiables à 100%, mais nous permettent, en considérant les trois, d'avoir une estimation de la qualité du clustering. Nous avons également pu observer les résultats des algorithmes en les affichant sous forme de graphes afin de visualiser les clusters par nous-même.

L'algorithme K-means démontre de très bonne performances pour des jeux de données non bruités, ayant des données bien séparées et des formes convexes plutôt arrondies. L'hétérogénéité de la densité des données n'influe pas sur ses performances. Cet algorithme reposant sur la détermination de "centres", qui forment les clusters, se révèle efficace sur des jeux de données très particuliers, et dont les clusters sont facilement identifiables, ce qui n'est pas forcément le cas pour des jeux de données réels. De plus, il faut lui renseigner le nombre de clusters attendus pour pouvoir obtenir des résultats, qui n'est souvent pas connu. Cependant, il possède un avantage concernant la rapidité d'exécution puisque son temps d'exécution, concernant nos jeux de données, restent inférieur à la demi-seconde.

Le deuxième algorithme, qui utilise la méthode agglomérative, démontre des performances similaires à la méthode k-means. En effet, son application obtient de bon résultats pour des jeux de données non bruités, dont les clusters sont bien séparés et convexes. De plus, pour cette méthode, il faut également passer en paramètre le nombre de clusters attendus, ce qui n'est normalement pas connu pour des applications réelles de clustering. En revanche, contrairement à k-means, cette méthode semble moins sensible à la forme des données, et permet d'identifier des clusters de formes allongés ou ovales. Cet algorithme permet donc de former des clusters dans des jeux de données un peu moins spécifiques. Il faudra cependant choisir un linkage approprié afin d'obtenir des résultats satisfaisants. On a vu que les linkages les plus efficaces ici étaient *ward* et *average*. D'autre part, cet algorithme a également l'avantage d'être rapide puisque son temps d'exécution reste, pour nos jeux de données, inférieur à 1 seconde.

Si les 2 algorithmes précédents étaient efficaces uniquement dans des cas convexes, un des principaux intérêts de DBSCAN est sa capacité à trouver des clusters *non convexes*. Il est également très efficace pour détecter le bruit dans les données. DBSCAN ne requiert pas le nombre de clusters recherché. En revanche, il sera compliqué à utiliser sur des jeux de données contenant des groupements à densité variables, ou des groupements mal séparés. Cela est dû au fait que l'algorithme va 'propager les clusters' d'élément en élément, tant que la densité est supérieure à celle sous-entendue par ses paramètres. Son temps d'exécution est très court, de quelques centièmes de seconde.

Pour pallier aux cas particuliers nous pouvons utiliser HDBSCAN qui propose des caractéristiques similaires, avec en plus une meilleure gestion du bruit, mais surtout de meilleurs résultats sur les données à densité variables, en modifiant la fonction de distance pour qu'elle soit défavorable au bruit et se basant sur des graphes Shared Nearest Neighbor. Ce traitement plus lourd lui donne un temps d'exécution bien plus lent que DBSCAN, allant d' $\frac{1}{4}$ de seconde à $\frac{1}{2}$ seconde. L'algorithme reste cependant rapide, avec une complexité $O(n \log(n))$.

Nous avons donc pu observer au travers du traitement de ces différents datasets quels algorithmes permettaient l'obtention du meilleur résultat selon le type de dataset à analyser. En effet, nous avons réalisé qu'il est important d'adapter notre choix de méthode selon les données. C'est cette expérience qui nous permettra dans le futur de faire ce choix sans avoir à essayer les différentes méthodes sur nos données. Nous avons également pu expérimenter la difficulté du choix des paramètres, notamment quand ils sont multiples, et essayer différentes méthodes pour trouver ces paramètres.