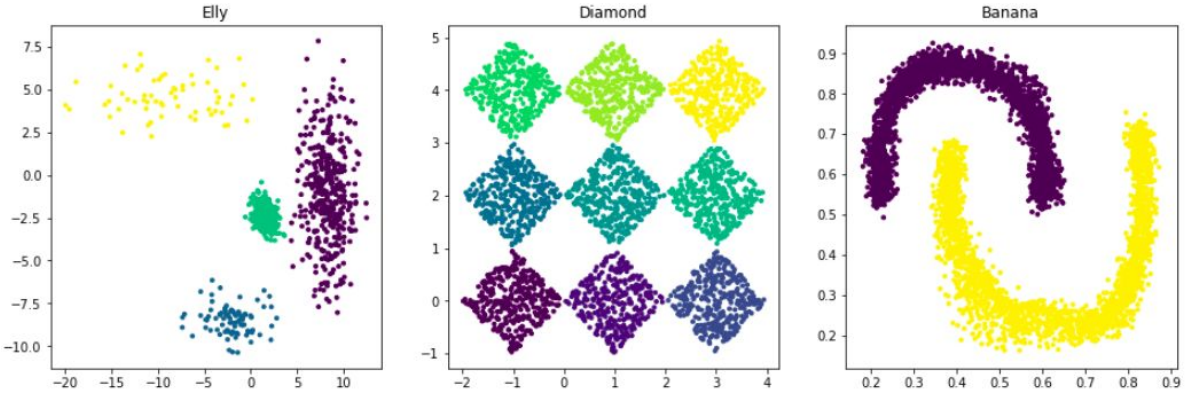
Compte-rendu de TP

Apprentissage non supervisé

Thomas CAZAUBON - 5SDBD_A2 - 2019/2020

I - Jeux de données

J'avais initialement sélectionné 5 jeux de données à utiliser avec les différents algorithmes étudiés dans ce TP. Finalement, il s'est avéré que 5 datasets différents ne faisaient qu'embrouiller mes résultats et alourdir l'étude des performances. J'ai donc décidé de ne garder au final que les 3 datasets qui me semblaient les plus pertinents. Les voici :



- **Elly** :
    - 4 clusters convexes de formes et densités variées.

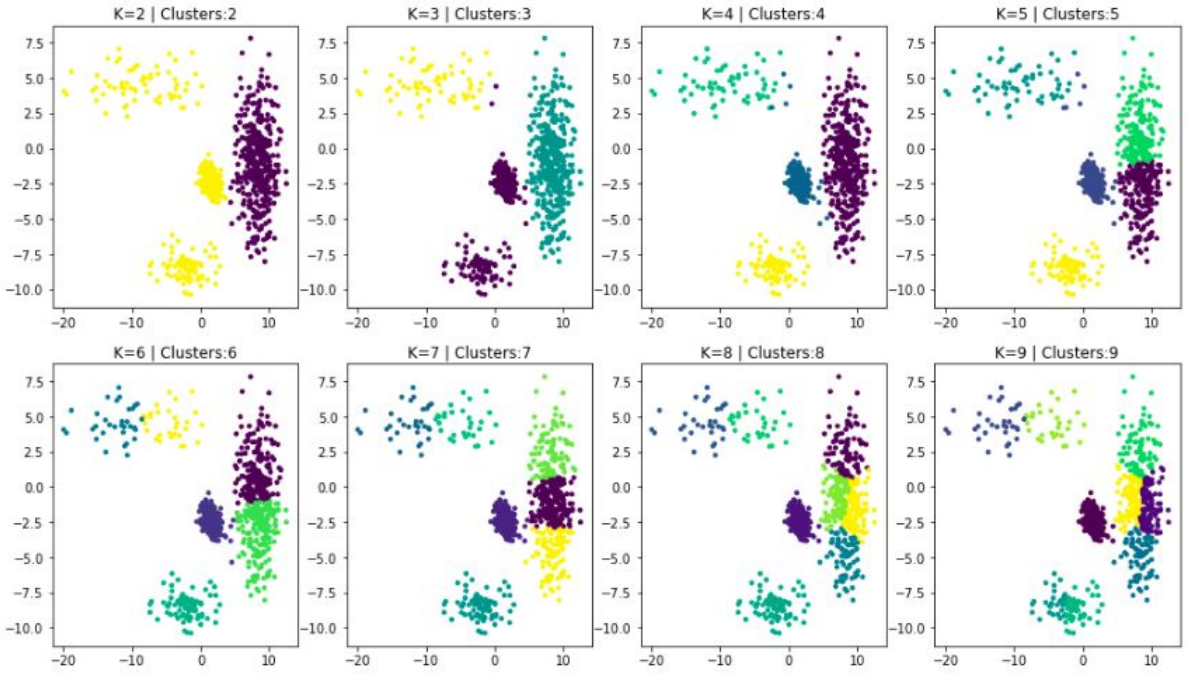
- **Diamond** :
    - 9 clusters convexes de formes et densités homogènes.

- **Banana** :
    - 2 clusters non convexes de formes et densités homogènes.

II - KMeans

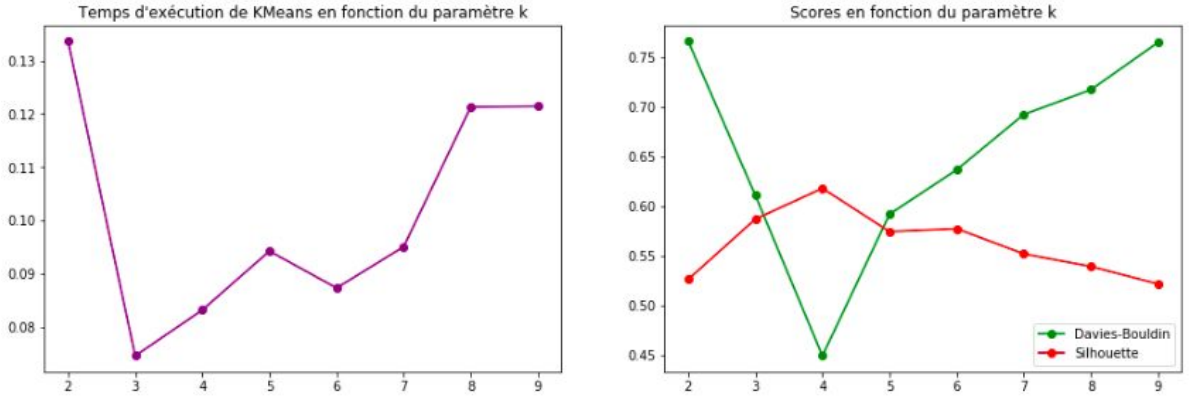
Le premier algorithme auquel nous allons nous intéresser est KMeans. Afin d'évaluer ce dernier on va simplement, dans un premier temps, faire varier le paramètre K et observer comment le clustering évolue sur nos différents datasets.

Commençons par **Elly** :



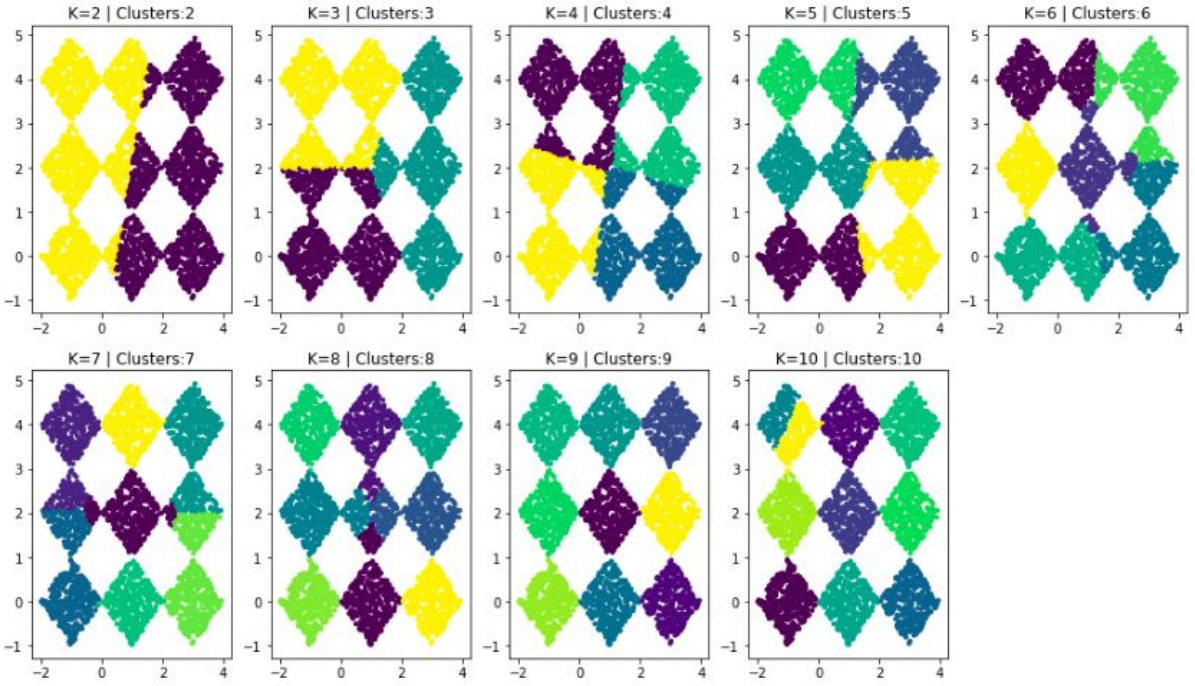
Application de KMeans sur Elly pour K de 2 à 9

K indiquant simplement le nombre de clusters à construire, les résultats obtenus ne sont pas étonnants, KMeans construit pour chaque valeur de K un nombre correspondant de clusters. Elly ne contient que 4 clusters, on peut constater de manière visuelle que le clustering pour K = 4 est le plus proche du dataset original. Cette affirmation est confirmée par le biais des métriques :



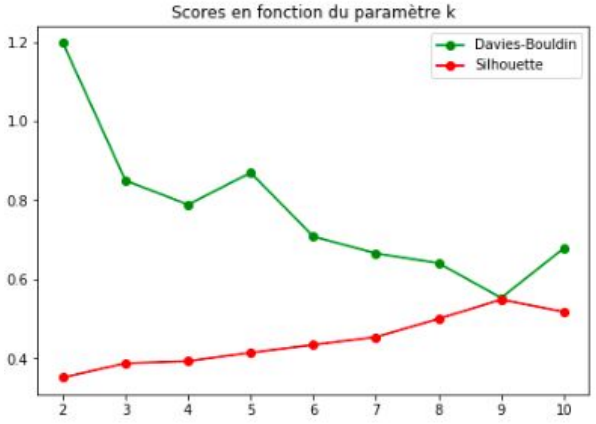
Temps d'exécution et scores en fonction de K sur Elly

Le temps d'exécution semble plus ou moins inversement proportionnel à K. Pour ce qui est des performances, on remarque bien un creux pour Davis-Bouldin (plus le score est faible, meilleur est le clustering) et un pic pour Silhouette pour K = 4. On voit cependant sur les différents graphiques que KMeans rencontre quelques difficultés pour les points les plus excentrés et qu'il a tendance à les attribuer au mauvais cluster (première limitation de KMeans). Nous allons donc tester sur **Diamond**, un jeu de données plus adapté pour cet algorithme.



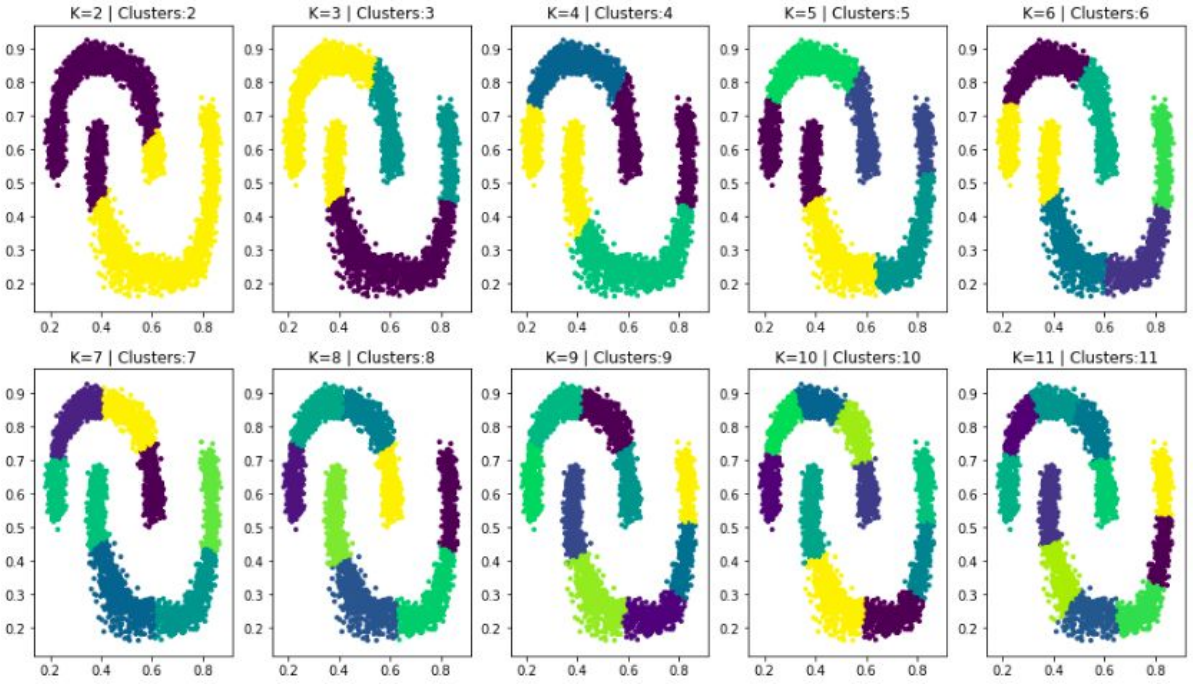
Application de KMeans sur Diamond pour K de 2 à 10

Cette fois, KMeans parvient à parfaitement reproduire les clusters d'origine (Pour K=9 bien-sûr), et ne commet aucune erreur.



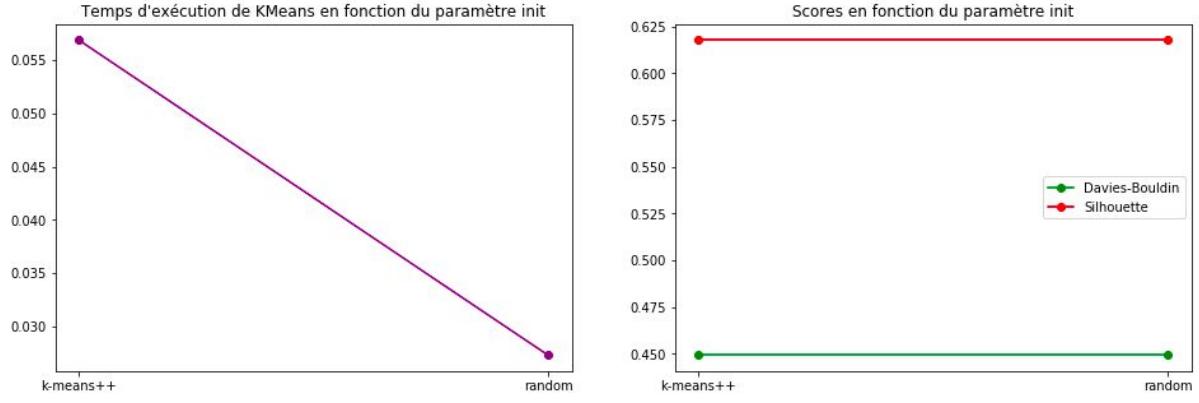
Scores en fonction de K sur Diamond

Les scores obtenus sont bien en accord avec l'observation. Nous allons maintenant montrer les limites de KMeans en utilisant un dataset contenant 2 clusters non convexes : **Banana**.



Application de KMeans sur Banana pour K de 2 à 11

On observe tout de suite ici que KMeans se plante complètement et est incapable de correctement séparer les deux clusters. En effet, KMeans n'est pas fait pour fonctionner avec des clusters non convexes, on observe donc une deuxième limite de cet algorithme. Nous ne présenterons pas ici les scores obtenus puisque les métriques utilisées ne sont pas non plus faites pour les formes non convexes et se montrent totalement incohérentes.



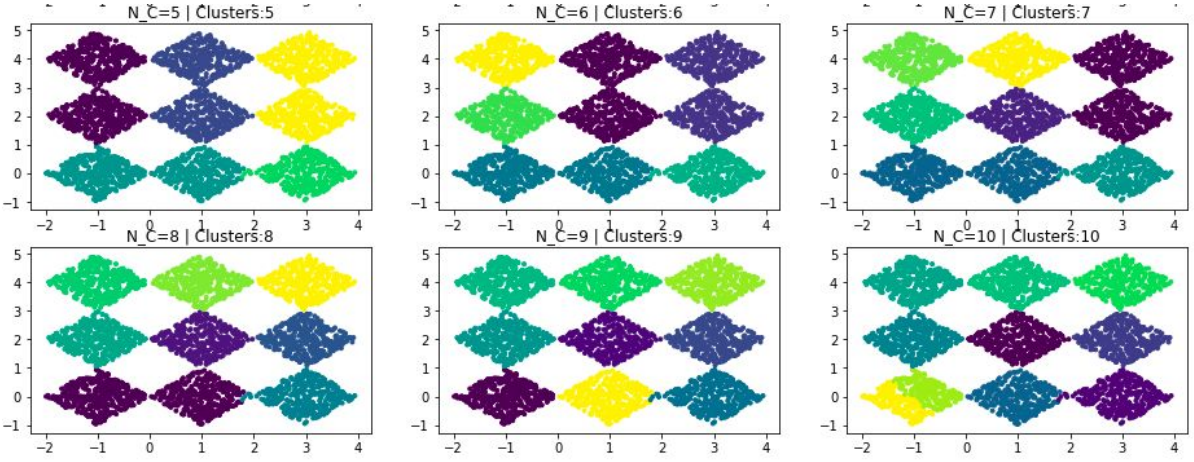
Temps d'exécution et scores en fonction de init sur Elly

En testant les 2 méthodes d'initialisation (k-means++ et random) on observe aucune différence sur la formation des clusters, en revanche, le temps d'exécution est légèrement réduit avec random. Petit détail : quand on fixe K à une grande valeur, chaque exécution ne donne pas le même résultat (en mode random comme k-means++).

BILAN : KMeans semble très à l'aise avec des formes géométriques régulières, de densités homogènes et convexes (type Diamond). En revanche, il présente des limites évidentes avec des clusters non convexes ou des formes et densités irrégulières.

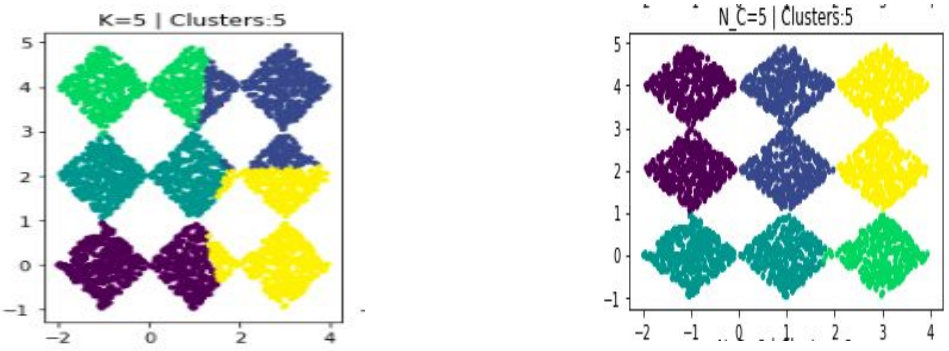
III - Clustering Agglomératif

Maintenant que nous avons pu tester KMeans et en voir les principales limites, nous allons nous pencher sur un nouvel algorithme : le clustering agglomératif.



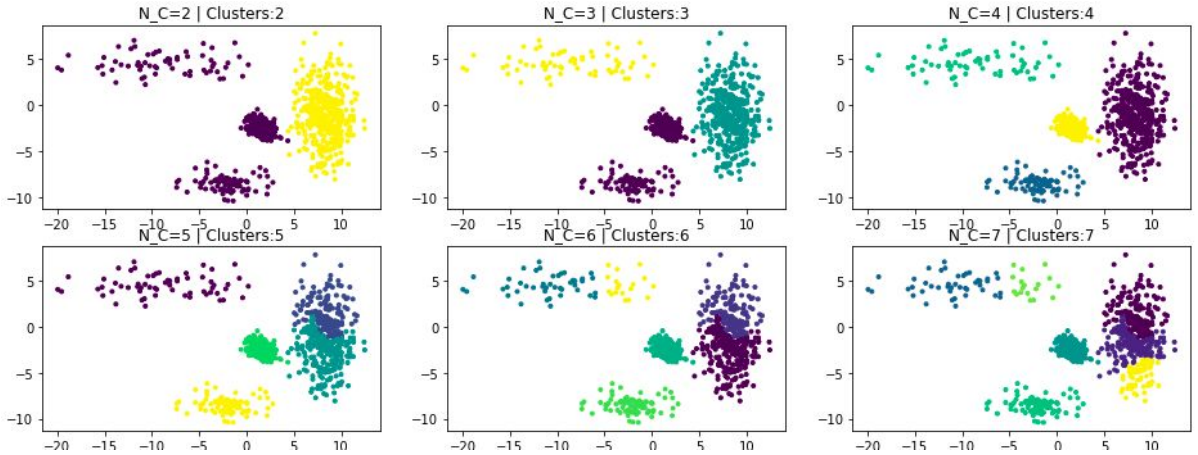
Application du Clustering Agglomératif sur Diamond pour K de 5 à 10

On constate ici avec surprise que cette méthode là se montre moins précise que KMeans sur le dataset **Diamond**. En effet, on observe la présence d'un léger débordement sur le cluster jaune pour K=9. En revanche, il est évident que cet algorithme parvient à mieux découper les clusters pour des K inférieurs à 9 :



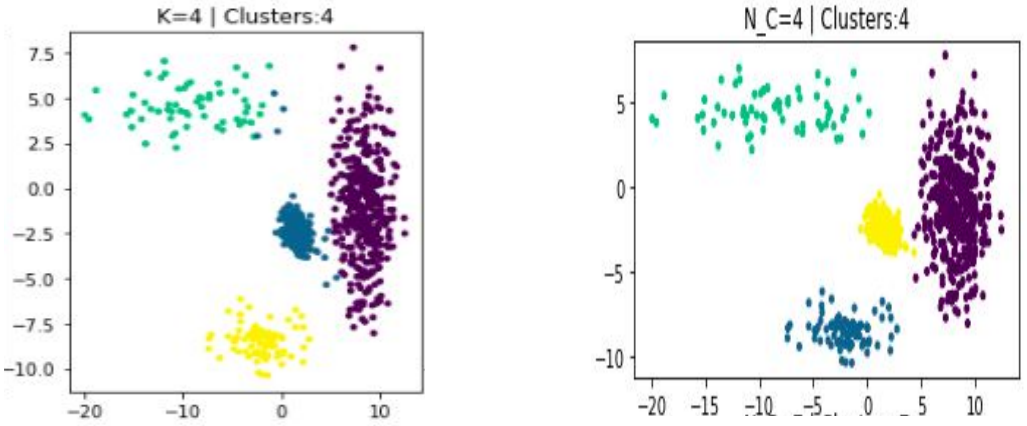
Comparaison d'un découpage avec KMeans (à gauche) et le Clustering Agglomératif (à droite)

Le découpage est plus logique avec le clustering agglomératif, qui préfère regrouper plusieurs clusters ensembles plutôt que de les scinder comme le fait KMeans. Essayons maintenant sur **Elly**.



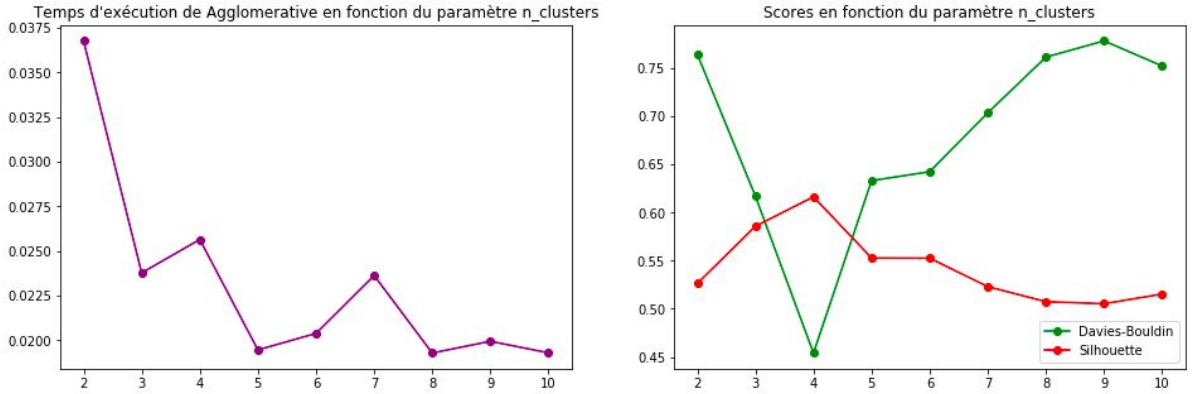
Application du Clustering Agglomératif sur Elly pour K de 2 à 7

Pour K=4, on voit que l'algorithme parvient mieux à découper les clusters que KMeans, voici un comparatif :



Comparaison d'un découpage avec KMeans (à gauche) et le Clustering Agglomératif (à droite)

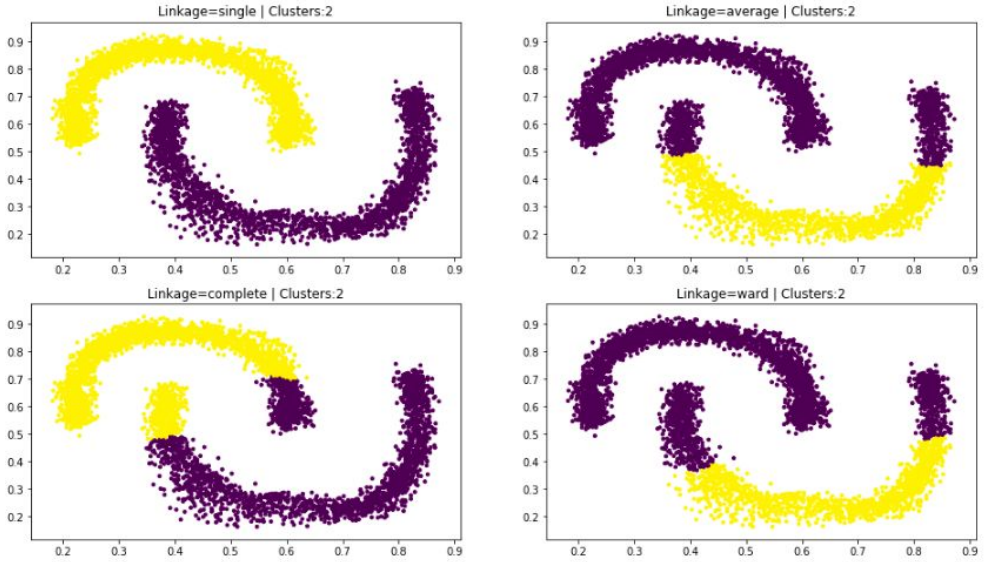
Le clustering agglomératif se montre moins sensible à la dispersion des points et la variété des formes et de densités.



Temps d'exécution et scores en fonction de K sur Elly

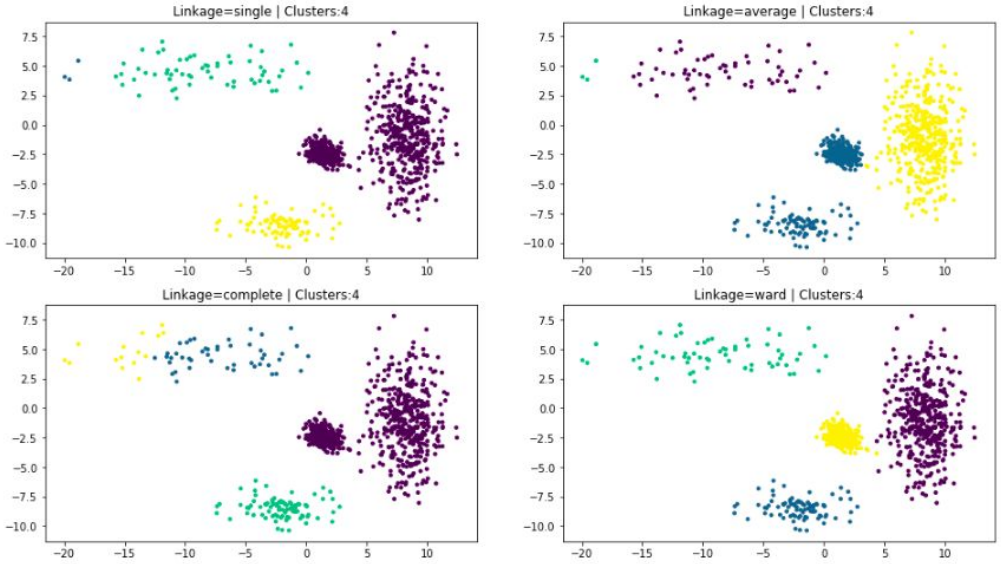
Le temps semble inversement proportionnel à K cette fois.

Nous allons maintenant mettre en évidence l'importance du choix du paramètre de linkage en prenant pour exemple un Banana et ses clusters non convexes et par conséquent non adaptés au clustering agglomératif. On fixe K à 2 et on teste les 4 modes de linkage différents :



Comparaison des différents modes de linkage sur Banana pour K=2

On remarque tout de suite que le mode de linkage single permet à notre algorithme de s'en sortir avec les honneurs, là où les autres modes ne lui permettent pas de correctement effectuer le clustering de Banana. Pourtant, si on effectue les mêmes tests sur **Elly** en fixant N à 4 :



Comparaison des différents modes de linkage sur Elly pour K=4

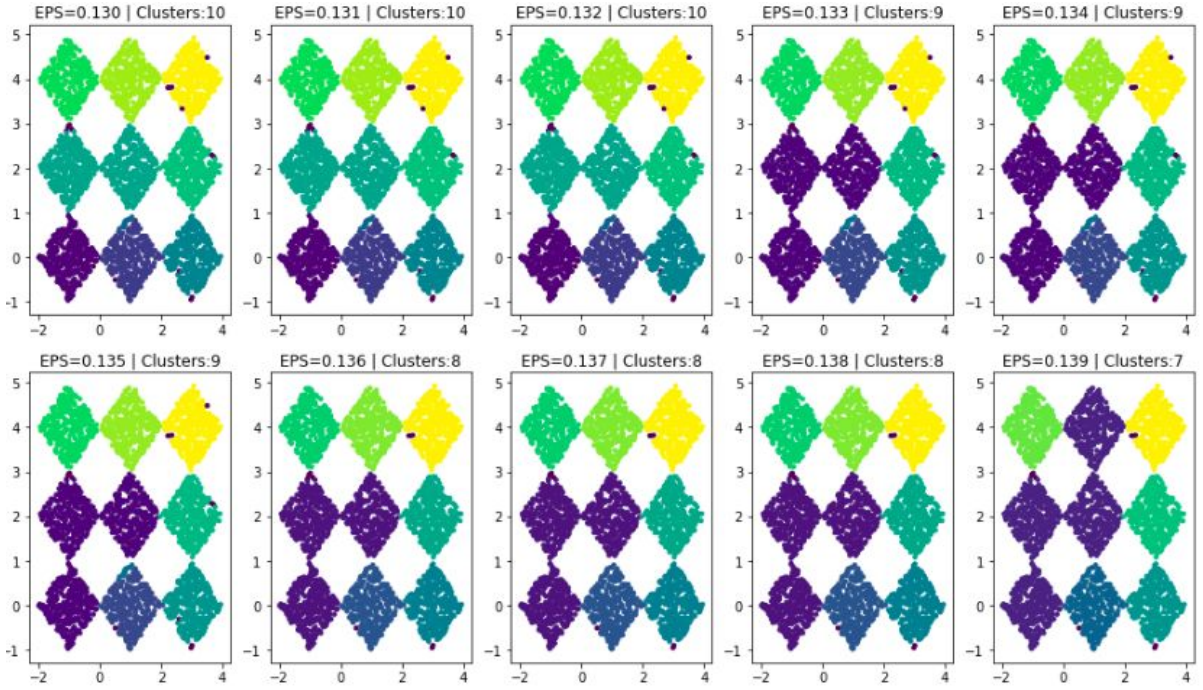
Les résultats sont bien différents ! C'est ward qui s'en sort le mieux, et de très loin (single est cependant le plus rapide). Ces expérimentations nous permettent de mettre en évidence l'importance de bien choisir ce paramètre en fonction du dataset étudié (Dans notre cas, ward pour des clusters convexes, et single pour des clusters non convexes) Il ne

faut pas se fier aux métriques qui affirment que le linkage single est le plus mauvais pour Banana, comme on l'a déjà indiqué, ces dernières ne sont pas faites pour évaluer les clusters non convexes.

BILAN : Le clustering agglomératif apporte une nouvelle flexibilité grâce à la possibilité d'adapter le mode de linkage aux données étudiées ce qui permet (dans les exemples vus) de s'en sortir admirablement bien avec des données non convexes et/ou hétérogènes. En revanche, sur des formes géométriques et densités homogènes, KMeans s'est montré plus performant. En termes de temps d'exécution, c'est encore une fois le clustering agglomératif qui l'emporte.

IV - DBSCAN

Maintenant que nous avons étudié 2 algorithmes bien adaptés aux clusters convexes, il est temps de nous intéresser à d'autres méthodes qui permettent également de nous attaquer à des formes non convexes : DBSCAN et HDBSCAN. Commençons par DBSCAN.

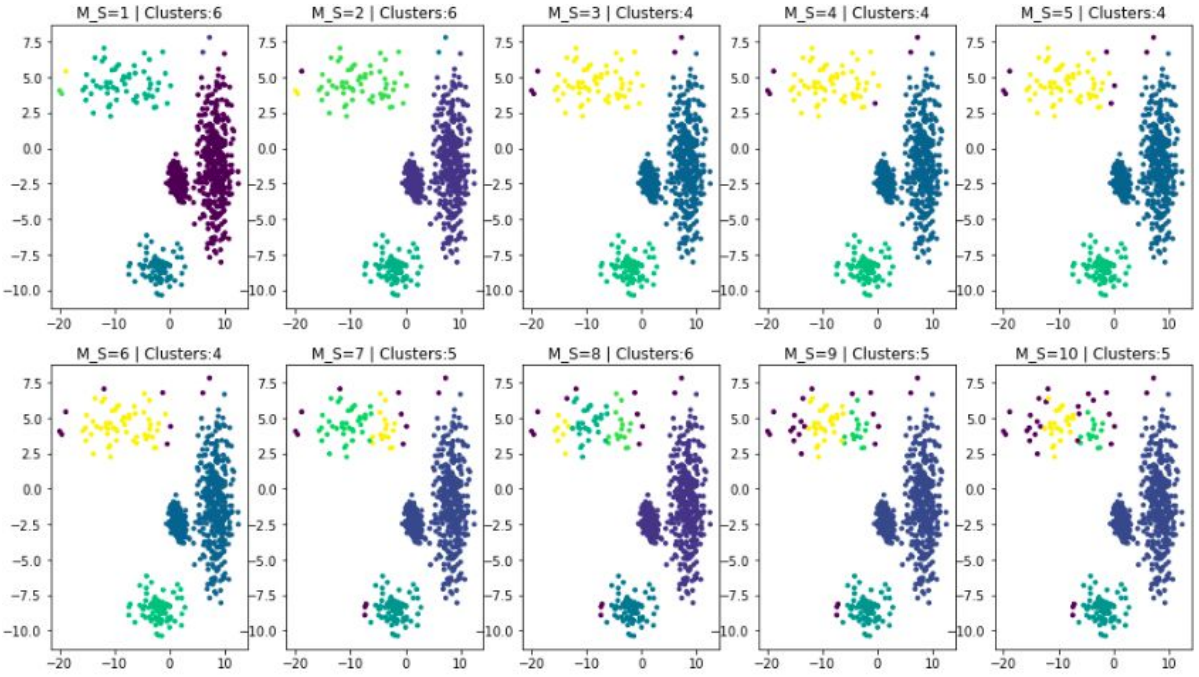


Application de DBSCAN sur Diamond pour eps de 0.130 à 0.139

DBSCAN s'en sort nettement moins bien que ses prédécesseurs sur ce type de données, on voit que certains points de clusters pourtant très éloignés sont rattachés à d'autres. Pour eps=0.134 on obtient bien 9 clusters mais ils sont assez éloignés de la réalité. Cet exemple mets aussi en avant un premier défaut de DBSCAN, il est nécessaire de fixer de la bonne manière les paramètres epsilon et min_samples ce qui est assez fastidieux. Il faut en effet procéder par étapes, en balayant large puis en affinant de plus en plus la

recherche des paramètres optimaux, pour au final ne pas forcément arriver à un résultat à la hauteur.

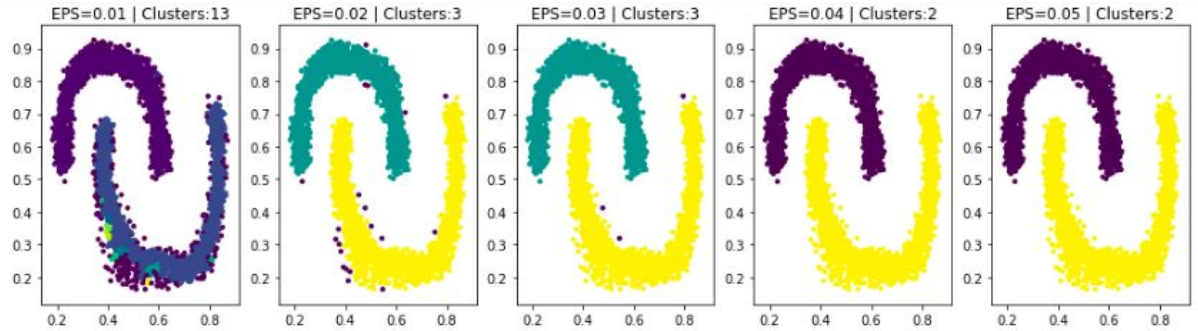
J'ai essayé de changer les valeurs de min_samples pour cet exemple mais je n'arrive au final qu'à retrouver un résultat identique au premier test avec une valeur de 5 (en fait la valeur par défaut de ce paramètre…). Je n'afficherai donc pas les résultats ici dans un souci de concision.



Application de DBSCAN sur Diamond pour min_samples de 1 à 10 (eps=1.7)

DBSCAN ne s'en sort pas franchement mieux sur **Elly**, après avoir fixé epsilon à 1.7, on fait varier les valeurs de min_samples mais aucune ne sort du lot en donnant un résultat réellement satisfaisant. **Petit aparté, DBSCAN ajoute une nouvelle classe représentée par la valeur -1 et qui regroupe tous les points considérés comme du bruit (ces points apparaissent en violet sur les extrémités des clusters). Même si cette distinction est une bonne chose de manière générale, ici elle vient perturber le clustering.**

Nous allons donc directement passer au seul intérêt de DBSCAN (Dans la limite de ce que j'ai pu constater bien-sûr !), à savoir son efficacité sur des clusters non convexes.



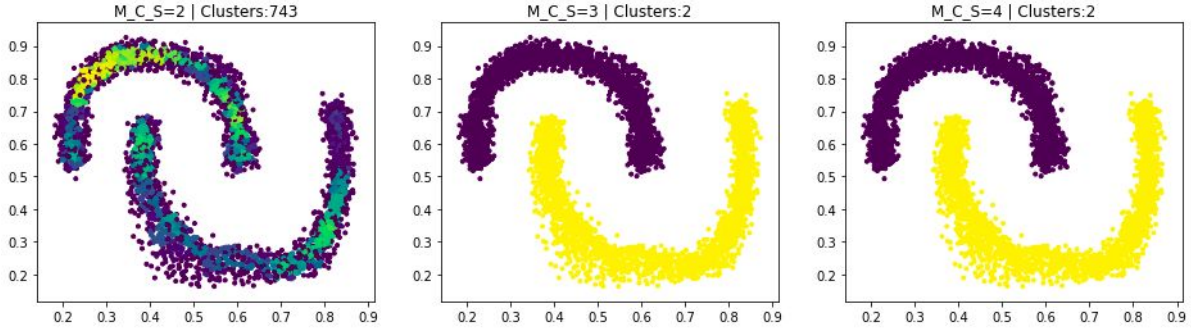
Application de DBSCAN sur Banana pour eps de 0.01 à 0.05

Sans surprise DBSCAN s'en sort très bien ici, il convient cependant de rappeler la difficulté de trouver les bons paramètres, la fenêtre est en effet en général très restreinte et il est facile de la rater, c'est donc une difficulté supplémentaire introduite par DBSCAN. Les variations de min_samples sont ici sans grand intérêt et on préfèrera généralement garder les valeurs par défaut.

BILAN : Je n'ai pas grand chose de plus à ajouter sur DBSCAN, nous allons donc directement conclure. De ce que nous avons pu voir, les performances de DBSCAN se montrent assez décevantes sur les clusters convexes comme Elly et Banana. En revanche, DBSCAN est assez puissant quand il s'agit de traiter des clusters non convexes. Les temps d'exécutions semblent assez similaires aux autres méthodes. Le gros problème de DBSCAN reste sa difficulté à configurer correctement pour obtenir des résultats adéquat, c'est là qu'HDBSCAN intervient et c'est ce que nous allons voir tout de suite.

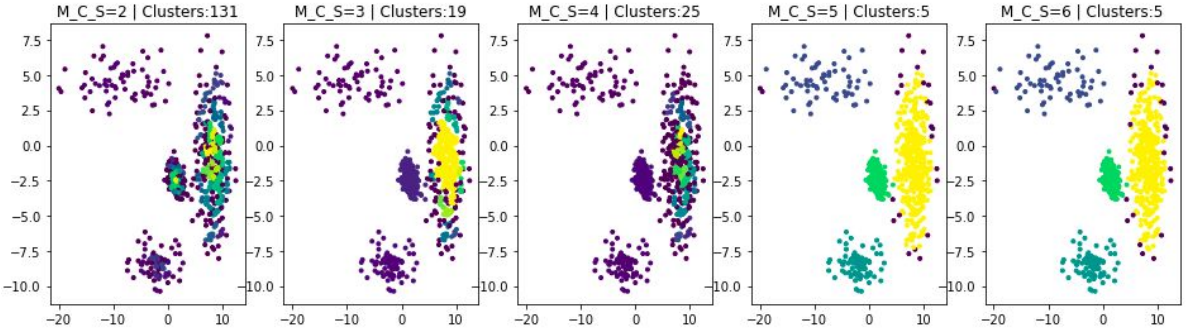
V - HDBSCAN

HDBSCAN est une variante de DBSCAN dont l'avantage principal réside dans le fait qu'il est "autoconfigurable" et que sa mise en application et par conséquent bien plus simple que pour DBSCAN. Commençons par **Banana** :



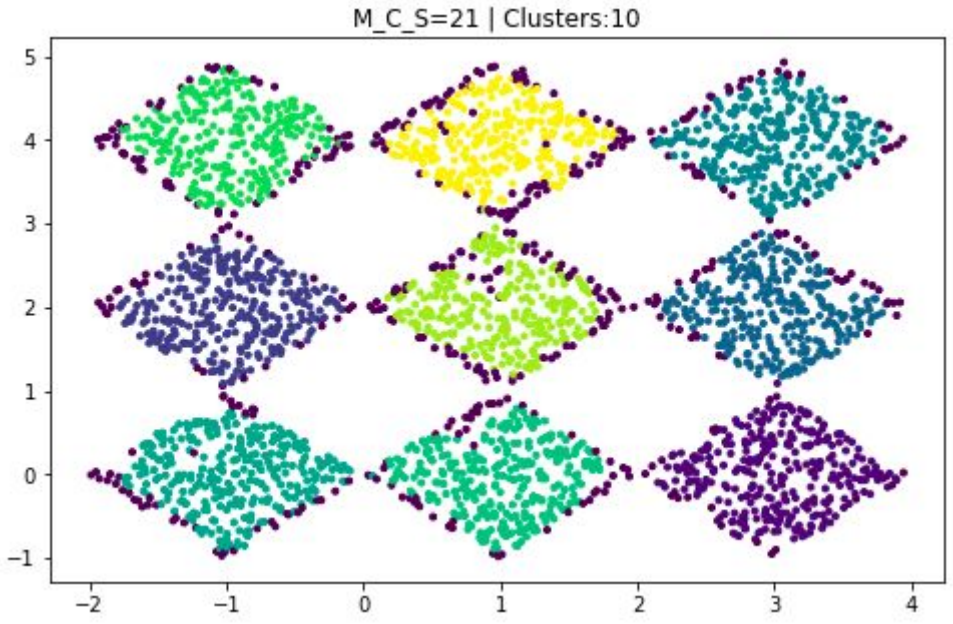
Application de HDBSCAN sur Banana pour minclustersize de 2 à 4

Rien de très surprenant ici, HDBSCAN donne des résultats similaires à DBSCAN mais en ne demandant qu'une configuration moindre. On ne s'attardera pas davantage sur cet exemple mais plutôt sur Elly qui réserve quelques bonnes surprises.



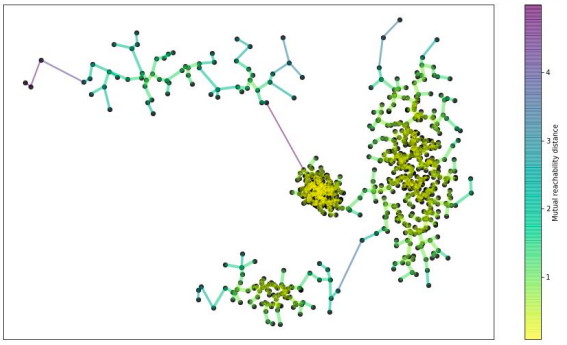
Application de HDBSCAN sur Elly pour minclustersize de 2 à 6

En effet, HDBSCAN nous montre ici des performances bien meilleures que DBSCAN et toujours avec une configuration bien plus légère. Cette fois, les 4 clusters sont reconnus comme il le faut, mais HDBSCAN classe certains des points extrêmes comme étant du bruit. C'est flagrant avec **Diamond** :

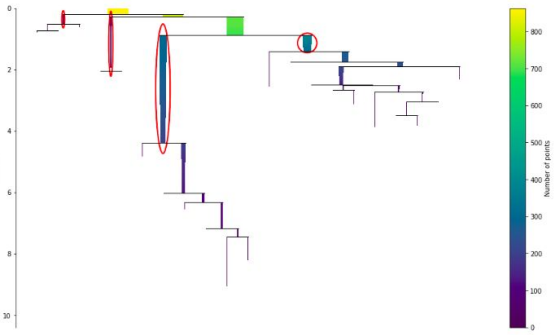


Application de HDBSCAN sur Diamond et visualisation du bruit

Les extrémités de chaque clusters sont perçues comme du bruit par HDBSCAN. On terminera cette partie en affichant l'arbre couvrant de poids minimal et l'arbre condensé sur le dataset **Elly**. Ils permettent de mettre en évidence la manière dont est construit le clustering.



Arbre couvrant de poids minimal sur Elly



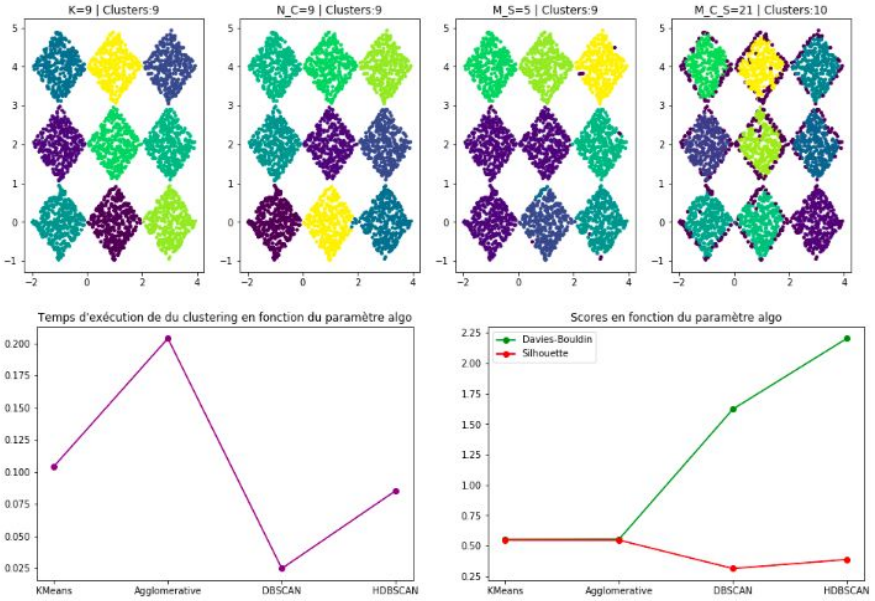
Arbre condensé sur Elly

BILAN : HDBSCAN est une version améliorée de DBSCAN se montrant plus facile à utiliser et surtout plus efficace à moindres efforts, l'inconvénient (ou pas d'ailleurs) étant qu'il a tendance à considérer une grande partie des points extrêmes des clusters comme étant du bruit, ce qui vient parfois perturber les résultats.

VI - SYNTHÈSE

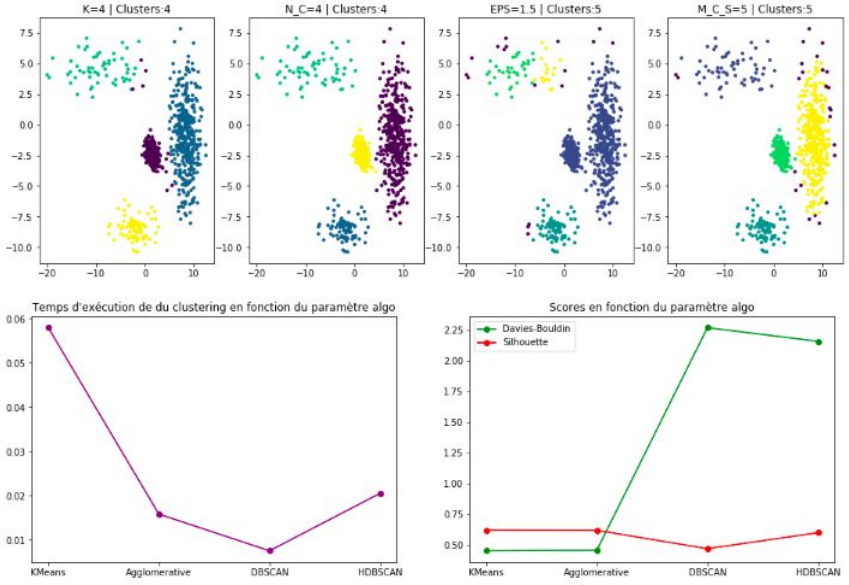
Nous allons à présent terminer cette étude par une petite synthèse comparative des 4 algorithmes que nous avons vus afin de mieux mettre en évidences leurs points forts et leurs faiblesses, en termes de temps et de performances. De gauche à droite à chaque fois : KMeans, Clustering Agglomératif, DBSCAN et HDBSCAN.

DIAMOND



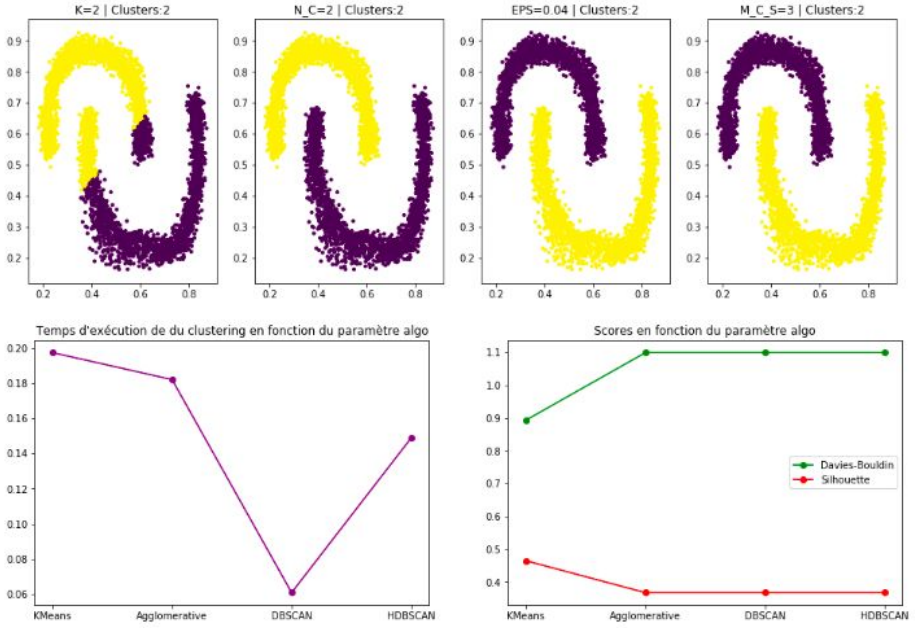
***Observations** : en termes de temps, c'est DBSCAN qui sort grand gagnant, en revanche, pour ce qui est des performances, KMeans et le clustering agglomératif s'en sortent bien (du fait qu'il ne reconnaissent pas de bruit), on notera de manière visuelle que KMeans s'en sort un tout petit mieux au fnal.*

ELLY



Observations *: DBSCAN est toujours le plus rapide, mais c'est le clustering agglomératif qui se montre le plus efficace (cela saute aux yeux sur l'affichage des résultats), même s'il partage le même score avec KMeans. DBSCAN et HDBSCAN sont une fois encore pénalisés par leur reconnaissance de bruit (A noter que DBSCAN est incapable de différencier les 2 clusters de droite).*

BANANA



Observations *: DBSCAN est encore une fois le plus rapide ! Cette fois on ne se fiera pas aux métriques puisqu'elles ne sont pas adaptées à ce genre de datasets, elles mettent cependant au même niveau les 3 derniers algorithmes, ce qui se confirme d'un coup d'oeil. Ici le clustering agglomératif a pu s'en sortir grâce au mode de linkage "single", mais c'est peut-être une exception. On préfèrera donc DBSCAN et HDBSCAN (on garde cependant en tête que DBSCAN est plus rapide !).*

BILAN GENERAL

Au vu des observations que l'on vient de faire, nous pouvons en tirer les conclusions suivantes :

- KMeans
  - Très performant sur des formes convexes simples et homogènes.
  - Nécessite de connaître à l'avance le nombre de clusters.
  - Long en temps d'exécution.
  - Sensible aux variations de formes et de densité.
  - Incapable de traiter les clusters non convexes.

- Clustering Agglomératif
  - Très performant sur tous types de formes même avec densités variables.
  - Flexible grâce au mode de linkage.
  - Peut s'en sortir sur des clusters non convexes…
  - … Mais n'est pas fait pour.
  - Nécessite de connaître à l'avance le nombre de clusters.
  - Relativement long en temps d'exécution.
- DBSCAN
  - Très rapide.
  - Relativement bien adapté aux formes non convexes ET convexes…
  - … En étant toutefois sensibles aux variations de formes et densités.
  - Configuration très fastidieuse.
  - Légèrement sensible au bruit.
- HDBSCAN
  - Très simple d'utilisation
  - Adapté aux formes non convexes ET convexes
  - Ne se soucie pas de l'hétérogénéité des données
  - Plus lent que DBSCAN
  - Très sensible au bruit

Pour synthétiser tout ça, dans le cas où l'application présente de fortes contraintes temps réel et des contraintes plus lâches en termes de fiabilité, on préfèrera DBSCAN qui s'en sort relativement bien avec des données homogènes. Si l'on est face à des formes convexes simples et homogènes, KMeans se montre très satisfaisant (attention au temps !). Le clustering agglomératif est une version plus polyvalente/flexible que KMeans tout en étant plus rapide, il fera l'affaire dans la plupart des cas, parfois même en présence de formes non convexes. Enfin HDBSCAN est un très bon compromis en termes de temps d'exécution et de fiabilité sur tous types de données, attention cependant à la présence de bruit !