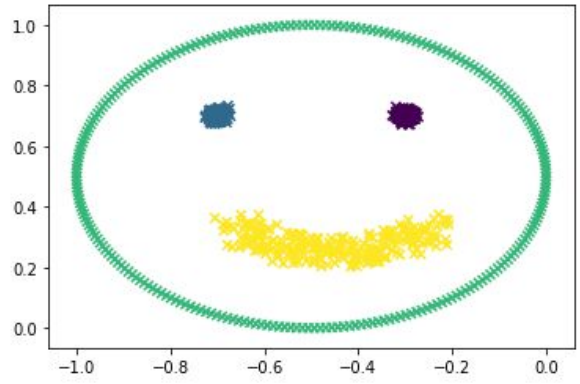
TP Apprentissage non-supervisé
Clustering

FEUILLET Laure - PLANTEC Maël

1. Jeux de données

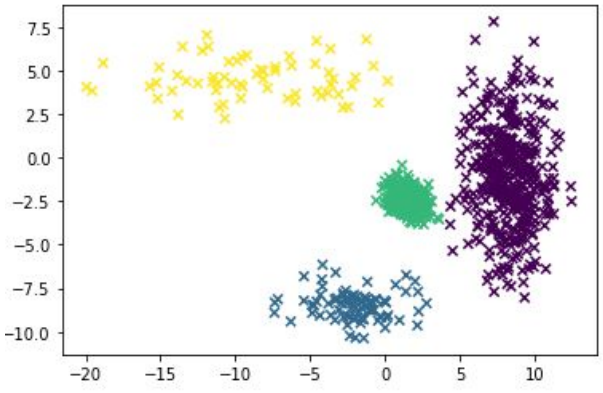
Nous avons choisi les deux datasets suivants :



Smile

>> Formes non-convexes
>> Formes bien séparées
>> Densité variable
>> Absence de données bruitées

https://github.com/deric/clustering-benchmark/blob/master/src/main/resources/datasets/artificial/smile1.arff



d2c4

>> Formes convexes
>> Formes mal séparées
>> Densités variables
>> Présence de données bruitées

https://github.com/deric/clustering-benchmark/blob/master/src/main/resources/datasets/artificial/2d-4c.arff

Pourquoi ? Il nous a semblé qu'ils se complétaient vraiment bien en terme de type de graphe obtenu (convexité, forme ...), ont des densités bien différentes et un "bruit" quasi nul pour l'un et important pour l'autre.

Dans la suite du TP nous allons appliquer les algorithmes suivants sur ces deux datasets :

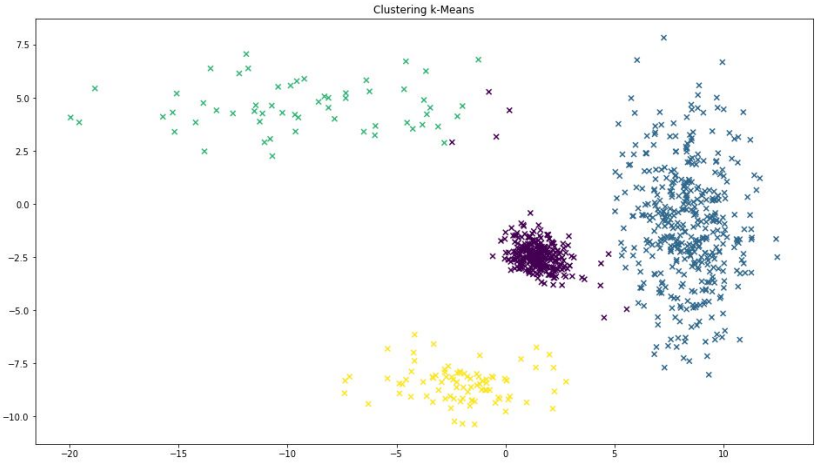
- clustering k-Means
- clustering agglomératif
- clustering DBSCAN
- clustering HDBSCAN

2. Clustering k-Means

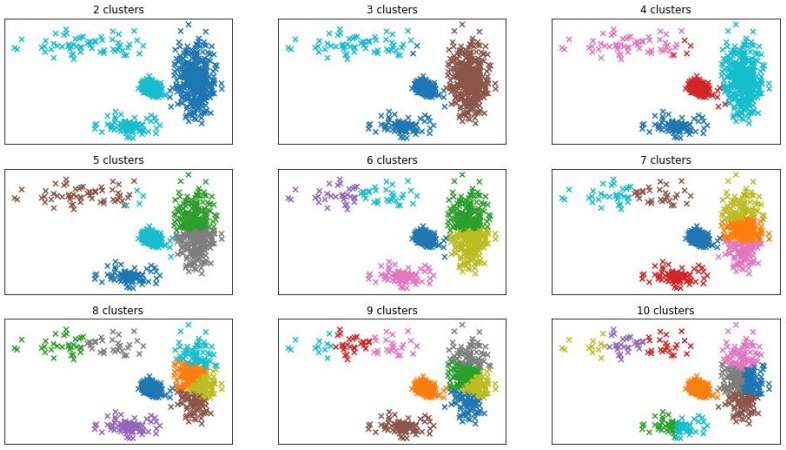
Commençons avec le clustering k-Means. Nous avons tout d'abord appliqué la méthode k-Means au dataset *d2c4*, car il n'a que des formes convexes qui lui sont plus adaptées. Nous le testerons ensuite sur le dataset *smile*.

a) Dataset *d2c4*

Puisque nous avons vu que ce dataset est composé de 4 clusters, commençons par réaliser un k-Means en spécifiant que l'on souhaite obtenir 4 clusters. L'allure générale de la solution ressemble globalement à celle du dataset initial.

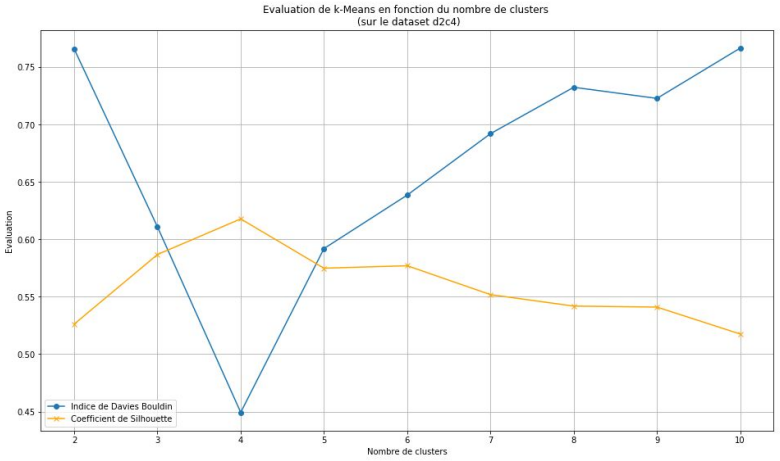


Dans le cas général d'un problème de clustering, on ne connaît pas a priori le nombre de clusters optimal. Faisons donc varier le nombre de clusters de 2 à 10.



Pour évaluer ces clusterings, il existe différents critères. Nous avons utilisé

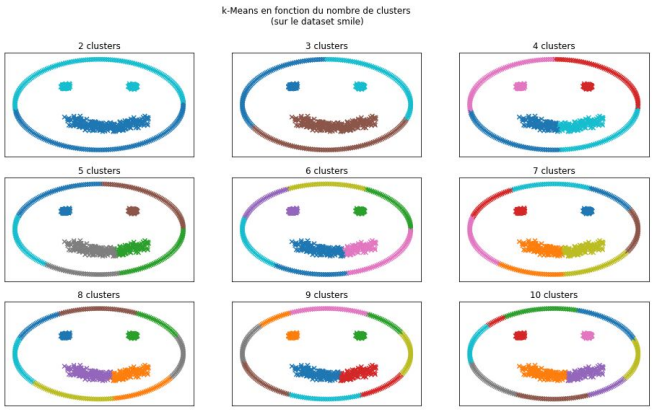
- l'indice de Davies Bouldin, qu'il convient de minimiser
- le coefficient de Silhouette, à maximiser.

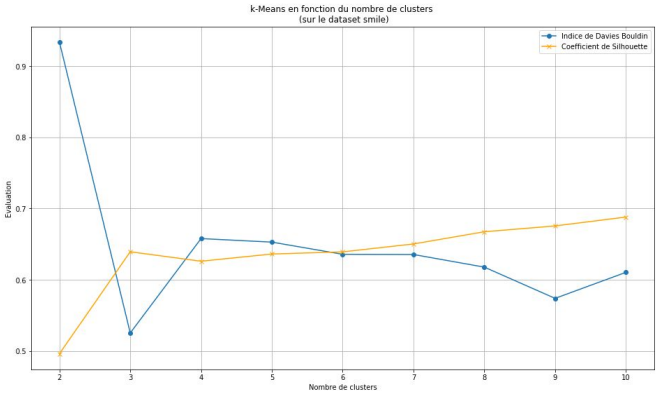


Si on affiche ces critères en fonction du nombre de clusters, il ressort que pour 4 clusters, on obtient le minimum des indices de Davies Bouldin et le maximum des coefficients de Silhouette. 4 est donc le nombre de clusters optimal pour ce dataset en se basant sur l'indice de Davies Bouldin et le critère de Silhouette. C'est parfait !

b) Dataset *smile*

Testons maintenant la méthode de k-Means sur notre deuxième dataset, smile, composé de formes non convexes.





On n'observe pas vraiment de pattern dans la variation de nombre de clusters avec cette méthode. Effectivement, en regardant l'allure des critères d'évaluation, il n'y a aucun clustering qui sort du lot. Cette méthode n'est visiblement pas adaptée à ce dataset comportant des formes non-convexes.

c) Conclusion

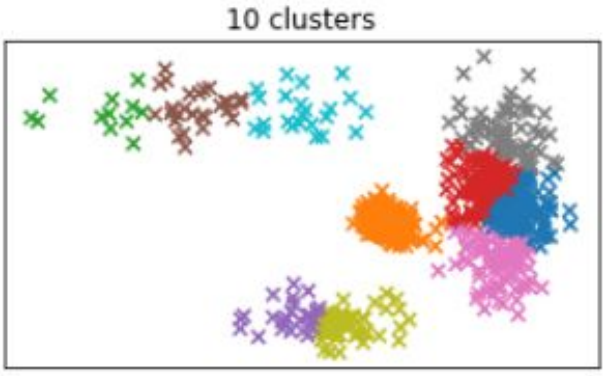
- **Pouvez-vous éviter de tester trop de valeurs différentes de k ?**

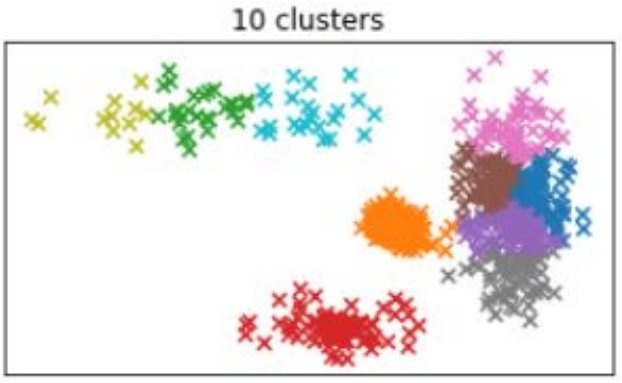
Puisque le nombre de clusters doit être fixé dans les paramètres du modèle, nous sommes obligés de tester des valeurs de k variées pour trouver le nombre de clusters qui correspond le mieux au dataset.
Au lieu de tester un grand nombre de valeurs différentes de k, on pourrait mettre en place une heuristique, réalisant des écarts entre deux valeurs de k à tester étant de plus en plus petits à mesure que l'on améliore certains indicateurs, comme l'indice de Davies-Bouldin ou le coefficient de Silhouette.

- **Retrouvez-vous une sensibilité de l'algorithme à l'initialisation ?**

On retrouve une sensibilité de l'algorithme à l'initialisation entre deux exécutions du même code. On observe ci-dessous par exemple que les clusters trouvés ne sont pas du tout le même pour k=10. Soit le

regroupement du bas est séparé en deux clusters et celui de droite en 4 clusters, soit le regroupement du bas forme un unique cluster et celui de droite se divise en 5 clusters.





- La méthode est-elle sensible à la nature des formes (cercle, rectangle, losange, non convexes, …) et à la densité des données ?

La comparaison des deux datasets permet de noter que la méthode de k-Means est sensible à la nature des formes. En effet, l'algorithme ne semble fonctionner que pour des formes convexes, comme pour le dataset *d2c4*. On voit bien que le clustering ne repère pas les formes que l'on souhaite sur *smile*.
Concernant la densité des données, k-Means y est sensible par nature, car l'algorithme consiste à utiliser la moyenne des points appartenant à un cluster comme référence. On observe par exemple sur le dataset *d2c4* que certains points ne sont pas attribués au bon cluster à cause de l'influence d'un autre cluster proche plus dense.
De plus, puisque tous les points sont inclus dans un cluster, cet algo est sensible au bruit et aux anomalies. On aurait pu essayer k-Means-- qui tolère un certain nombre d'anomalies.
Pour des formes mal séparées, mais de même densité, comme dans le dataset *d2c4*, l'algorithme arrive plutôt bien à différencier les clusters.

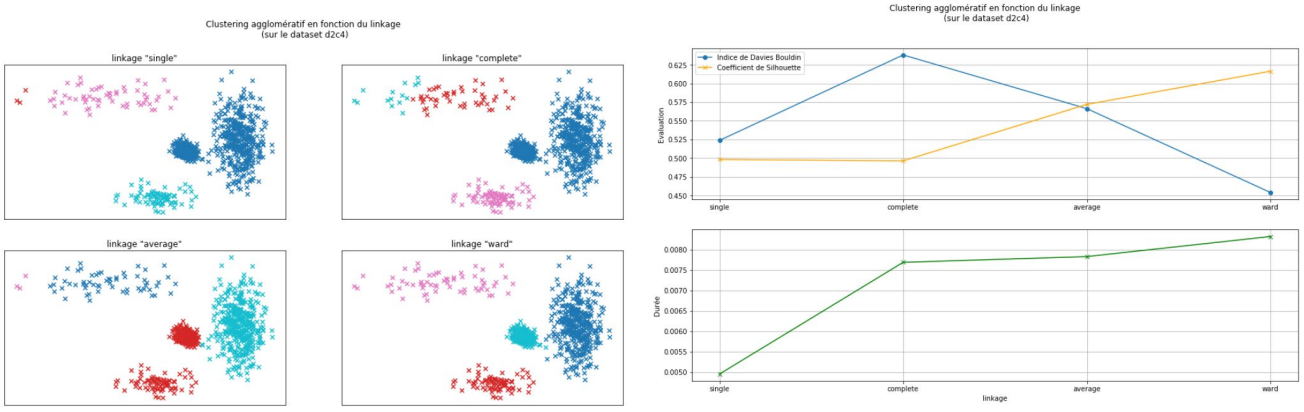
3. Clustering agglomératif

Nous allons maintenant tester le clustering agglomératif. Tout comme pour le clustering k-Means, il faut passer en paramètre le nombre de clusters attendus. Le clustering agglomératif consiste à fusionner chaque point avec le cluster le plus proche. Cela fait donc intervenir une notion de distance. Nous allons dans un premier temps utiliser uniquement la distance euclidienne. Il existe aussi différentes possibilités de combiner les clusters entre eux. Nous allons faire varier ce critère via l'un des paramètres de l'ago :

```
linkage : {"ward", "complete", "average", "single"},
optional (default="ward")
```

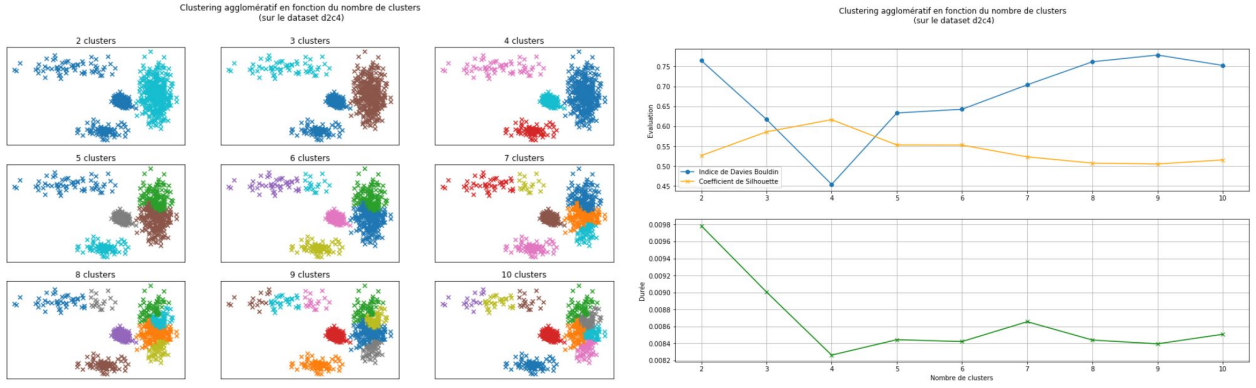
a) Dataset *d2c4*

Tout d'abord nous avons fait varier le paramètre *linkage* du clustering agglomératif sur le dataset d2c4. Nous avons précisé un nombre de clusters égal à 4, qui est la valeur que nous supposons être la plus adaptée.



Les critères d'évaluation nous indiquent que le linkage le plus adapté est "*ward*", ce que l'on peut confirmer visuellement. Cela signifie que les clusters sont fusionnés de sorte à minimiser la variance des clusters à fusionner.

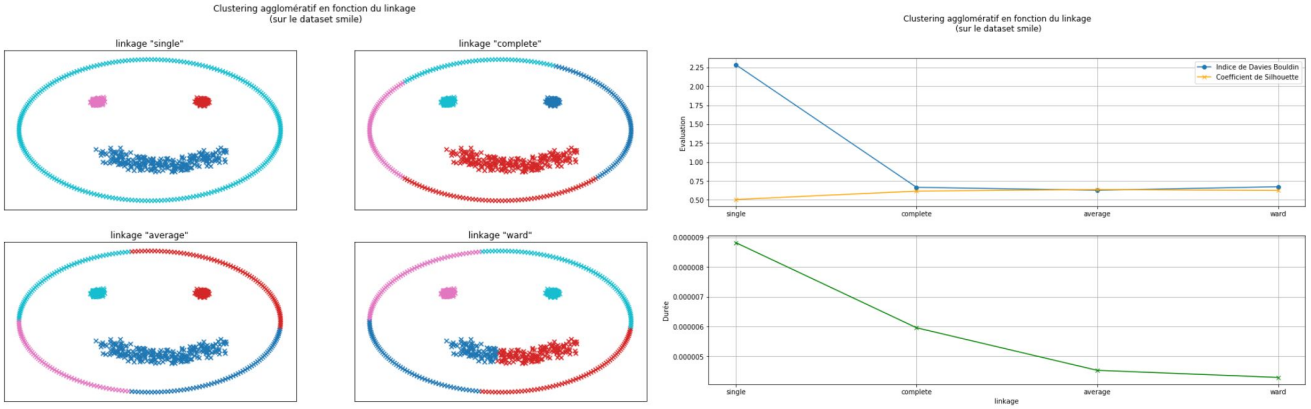
En gardant le linkage "*ward*", faisons maintenant varier le nombre de clusters passé en paramètre.



D'après les deux critères d'évaluation choisis (Davies-Bouldin et Silhouette), un nombre de 4 clusters apparaît comme le plus adapté pour le dataset *d2c4*. Parfait encore !

b) Dataset *smile*

Tout comme pour le dataset *d2c4*, commençons par comparer les différents *linkage* pour trouver le plus adapté au dataset *smile*.

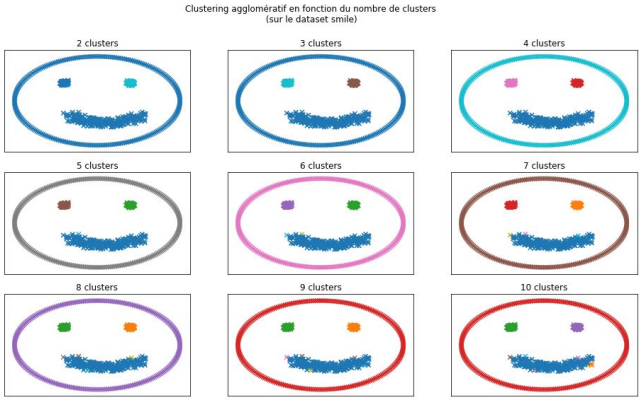


Visuellement, on observe que le *linkage=single* est le plus adapté. Cependant, les indicateurs ne nous montrent pas du tout ce résultat. Regardons plus en détails ces indicateurs pour comprendre d'où cela peut venir.

- L'**indice de Davies-Bouldin** se base sur les distances des centres des clusters. Pour des formes non-convexes, les centres des clusters n'appartiennent pas à leur cluster, et peuvent même appartenir à un autre, ce qui perturbe cet indicateur.
- Le **coefficient de Silhouette** représente la distance moyenne entre tous les points d'un cluster, et la moyenne des distance aux points des autres clusters. Puisque certaines formes de ce cluster sont non convexes, les points d'un cluster peuvent tout à fait être plus proches des points d'un autre cluster que de celui d'appartenance. Par exemple, la partie en haut à gauche du cercle (contour du visage) est plus proche du cluster de l'oeil que des autres points du cercle.

Ces deux indicateurs ne sont donc visiblement pas adaptés à un dataset comportant des formes non convexes.

Conservons *linkage=ward* pour la suite.



Puisque les indicateurs ne sont pas adaptés aux formes non-convexes, il ne sert à rien de les utiliser pour trouver le nombre de clusters le plus adéquat.
Visuellement, on remarque qu'à partir de 4 clusters, on retrouve bien la répartition attendue. Au-delà l'algorithme ajoute uniquement des clusters composés d'un point aux extrémités de la bouche.

c) Conclusion

- **Pouvez-vous éviter de tester trop de valeurs différentes de k?**

Puisque le nombre de clusters doit être fixé dans les paramètres du modèle, nous sommes obligés de tester des valeurs de k variées pour trouver le nombre de clusters qui correspond le mieux au dataset.
Au lieu de tester un grand nombre de valeurs différentes de k, on pourrait mettre en place une heuristique, réalisant des écarts entre deux valeurs de k à tester étant de plus en plus petits à mesure que l'on améliore certains indicateurs, comme l'indice de Davies-Bouldin ou le coefficient de Silhouette si le dataset est composé de formes convexes.

- Quel est l'impact des différentes combinaisons des clusters ?

Les différentes combinaisons de clusters permettent de choisir la grandeur à minimiser lors de la fusion de deux clusters due à la méthode de clustering agglomératif. Ce paramètre influe donc grandement sur les clusters trouvés.

- *ward* : variance entre les clusters à fusionner.
- *average* : distance moyenne entre les points des deux clusters à fusionner.
- *complete* : distance maximale entre les points des deux clusters à fusionner.
- *single* : distance minimale entre les points des deux clusters à fusionner.

Nous avons remarqué que *linkage=single* est particulièrement adapté aux formes non convexes, comme vu précédemment avec le dataset *smile*. Pour le dataset plus classique *d2c4*, c'est *linkage=ward* qui a donné les meilleurs résultats.

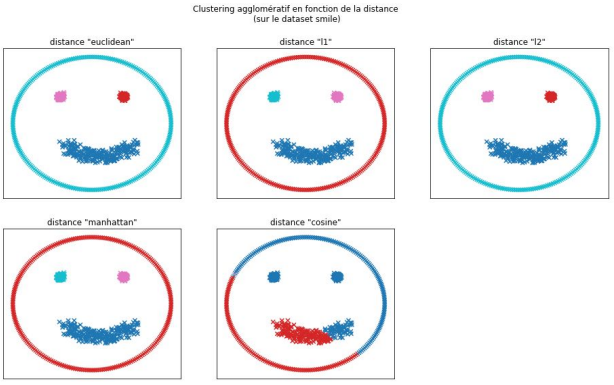
- La méthode est-elle sensible à la nature des formes (cercle, rectangle, losange, non convexes, ...) et à la densité des données ?

D'après l'application du clustering agglomératif sur les datasets *d2c4* et *smile*, nous pouvons en déduire que cette méthode n'est pas sensible à la nature des formes. Si on paramètre bien le modèle, notamment le *linkage*, l'algorithme est capable de distinguer parfaitement des cercles, des courbes, des patatoïdes...

La méthode est également robuste à la densité des données. Sur le dataset *d2c4* qui possède quatre formes de deux densités différentes, les clusters ont bien été identifiés. Les groupes de densité plus faibles ne sont pas "attirés" par ceux de densité plus forte. Cela peut s'expliquer par les méthodes de fusion qui prennent en compte des grandeurs représentatives de tout le cluster, comme la distance minimale entre tous les points, et pas le nombre de points du cluster. Ainsi la densité est bien prise en compte.

d) Options

- La méthode est-elle sensible à la métrique de distance (euclidienne, manhattan, ...) ?



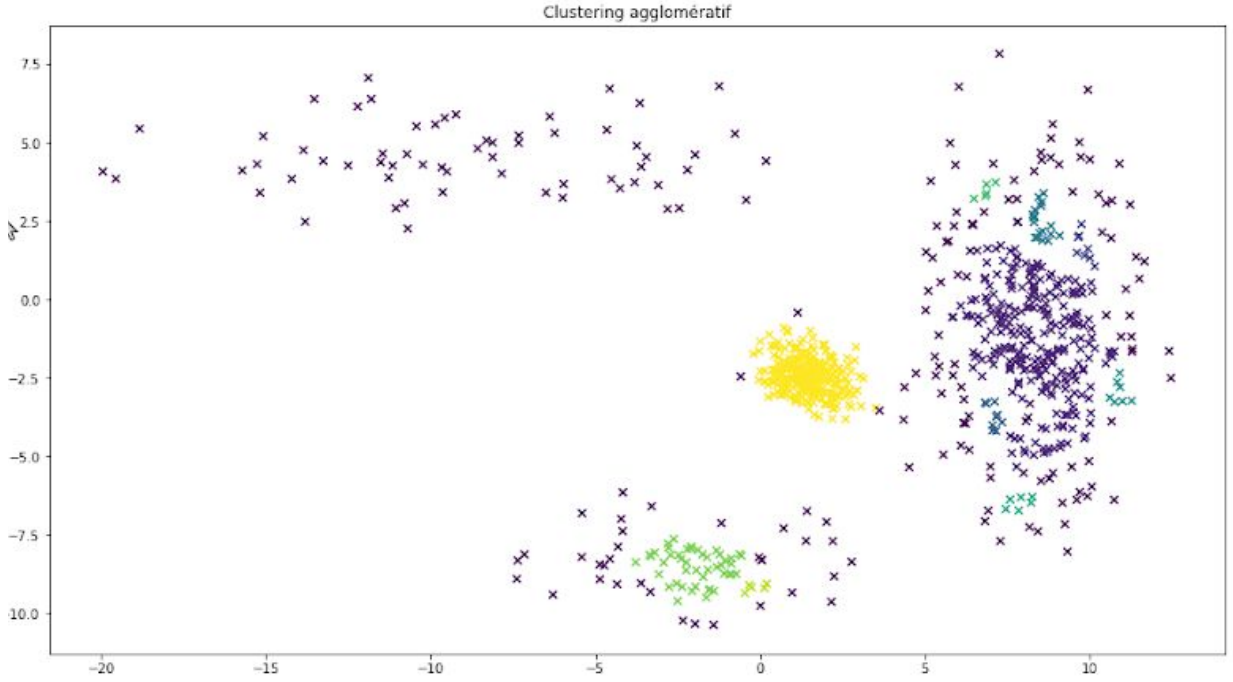
Pour tester si la méthode de clustering agglomératif est sensible à la distance utilisée, nous avons fait varier la distance utilisée pour le clustering du dataset *smile* (il n'est pas possible de le faire sur *d2c4*, car il utilise le linkage *ward* qui ne permet que la distance euclidienne).

On observe que seule la distance *cosine* change la solution. Il faut donc également faire attention à la métrique de distance utilisée.

4. Clustering DBSCAN

a) Dataset "*d2c4*"

Nous avons réalisé un premier jet avec les valeurs par défaut du clustering DBSCAN, en voici le résultat :



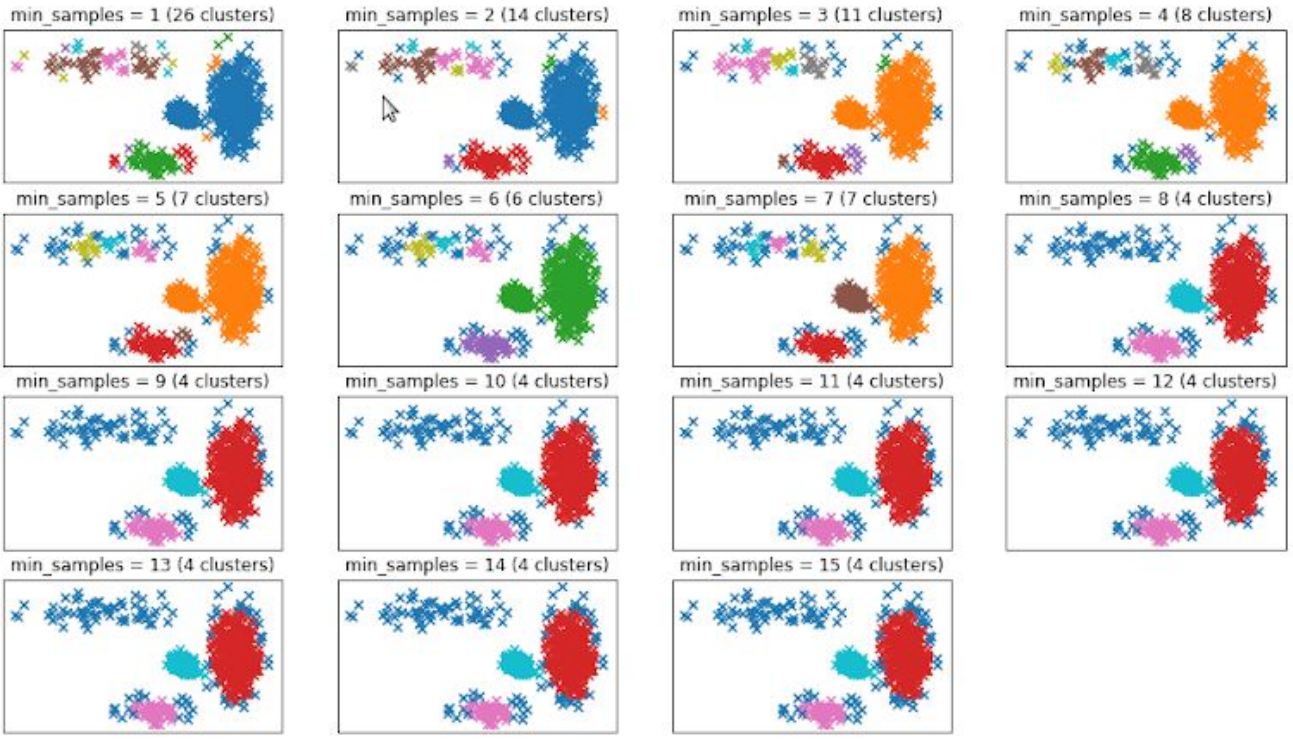
Il y a 11 clusters dans ce graphe. La valeur que nous attendions est 4, il va donc falloir jouer avec les paramètres suivants pour affiner le processus de clustering.

eps : float
Ce premier paramètre représente la distance maximale entre deux points pour qu'ils soient considérés dans le voisinage de l'autre. C'est apparement le paramètre le plus important de DBSCAN à définir pour un dataset particulier et la fonction de distance dont on parlera plus tard.

min_samples : int
Le nombre de points (ou poids total) d'un voisinage pour qu'un point soit considéré comme un coeur de cluster.

Variation de ces paramètres :
Nous avons tout d'abord chercher à définir le paramètre "eps". Ainsi, un peu à taton et grâce à une boucle, uniquement en regardant les allures des résultats, nous nous sommes arrêtés sur une valeur de eps égale à 1.1 pour ce dataset.

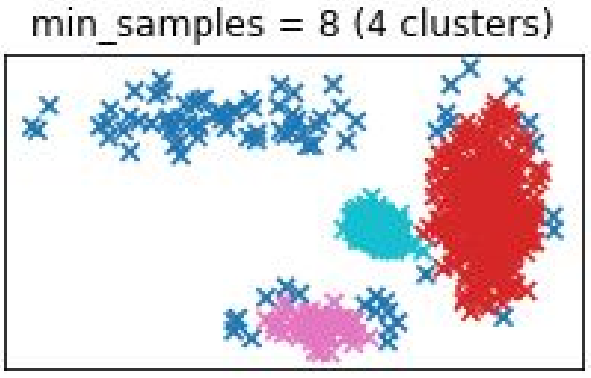
Puis, pour min_samples, nous avons décidé, au vu des graphes pour min_samples = 1 et min_samples = 15, de faire des tests pour des valeurs plus précises ce cet intervalle [1,15]. Nous affichons le nombre de clusters défini pour chaque graphe, donc pour chaque valeur de min_samples.



La valeur qui sort du lot est min_samples = 8. Ainsi, nous avons :

- **eps = 1.1**
- **min_samples = 8**

Avec ces paramètres nous trouvons bien 4 clusters dans le résultat.



On pourrait s'arrêter là, hélas, si on regarde bien le graphe ci-dessus, on se rend compte qu'une des classes (la bleue) a des points disséminés dans les contours des autres. Ces points qui semblent être le bruit de certains clusters sont cependant regroupés en haut à gauche en un clusters. **Bruit ou cluster ?**
→ Dans l'attribut "labels_" du modèle dbscan on peut voir les labels des points et donc à quels clusters ils sont affectés. Les points considérés comme du bruit ont le label **-1.** Or si on fait :

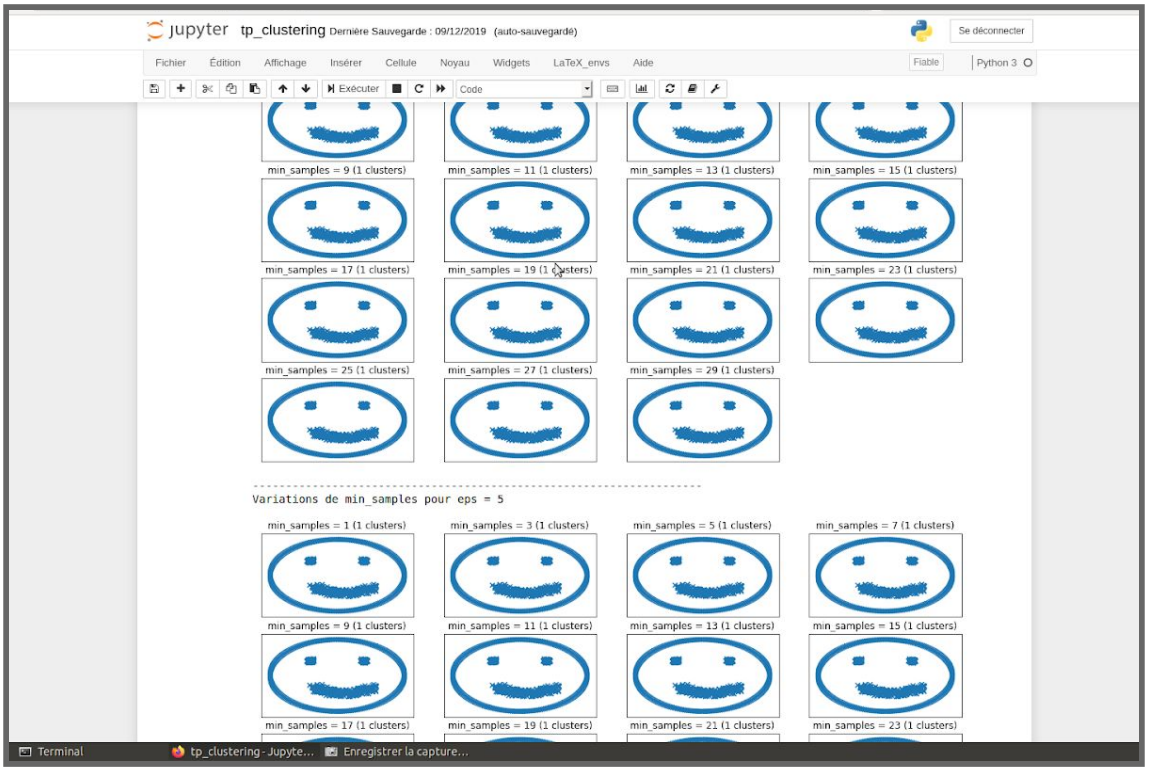
```
print(np.unique(dbscan_d2c4.labels_))
```

On obtient :

```
[-1  0  1  2]
```

Un de nos clusters n'en est pas un, c'est juste un groupe de tous les points considérés comme du bruitage.

b) Dataset "Smile"



⇒ Pour le dataset smile, nous avons essayé toutes les combinaisons possibles de eps et min_samples mais en vain.

- La méthode est-elle sensible à la nature des formes (cercle, rectangle, losange, non convexes, …) et à la densité des données ?

La densité des données semble à première vue être un problème (partie a) mais peut-être n'avons nous pas assez joué avec les paramètres de DBSCAN. Dans le dataset smile qui a une densité uniforme entre ses classes c'est le problème de la forme qui survient :
Cette méthode est extrêmement sensible à la forme. En effet, comme le montre la partie b), la forme convexe brise le processus de clustering fait par DBSCAN.

- Le bruit dans les données est-il bien identifié ?

Non, du moins avec nos paramètres et sur le dataset d2c4, il est net que si une part du bruit est bien identifiée, ce qui est sensé être une classe entière est aussi considéré comme du bruit, et cela ne va pas.

- Optionnel : La méthode est-elle sensible à la métrique de distance (euclidienne, manhattan, …) ?

Mis à part pour "cosine" (qui ne trouve qu'un seul cluster) nous n'avons vu aucune différence dans les résultats en utilisant des métriques de distance différentes.

5. Clustering HDBSCAN

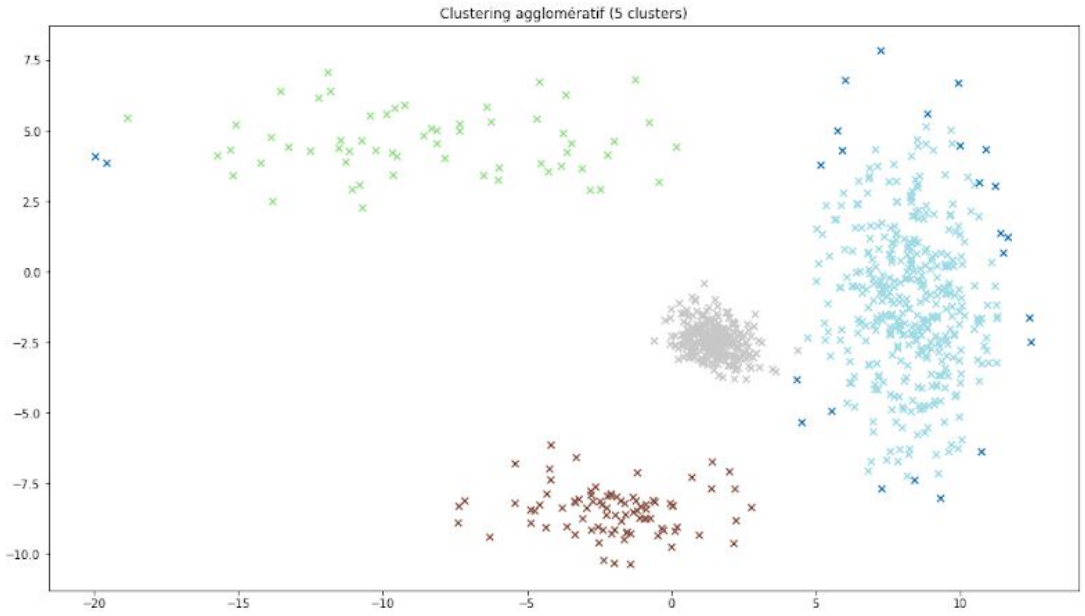
HDBSCAN est une **extension de DBSCAN** vu précédement. La différence est dans le "H" de hiérarchique. C'est donc un algorithme de **clustering hiérarchique** qui va ensuite utiliser une technique basée sur la stabilité des clusters pour extraire ces derniers. Cette technique est décrite ici https://link.springer.com/article/10.1007/s10618-013-0311-4 et est appelée "**flat cluster extraction**", nous en verrons les utilisations plus tard dans cette partie.

Ci-dessous, les **paramètres** de notre algorithme HDBSCAN :

```
HDBSCAN(algorithm='best', alpha=1.0, approx_min_span_tree=True,
    gen_min_span_tree=True, leaf_size=40, memory=Memory(cachedir=None),
    metric='euclidean', min_cluster_size=5, min_samples=None, p=None)
```

a) Dataset D2C4

Commençons par entrainer un modèle HDBSCAN sans modifier les paramètres. et en utilisant le dataset avec des formes simples (`hdbscan_d2c4 = hdbscan.HDBSCAN()`).

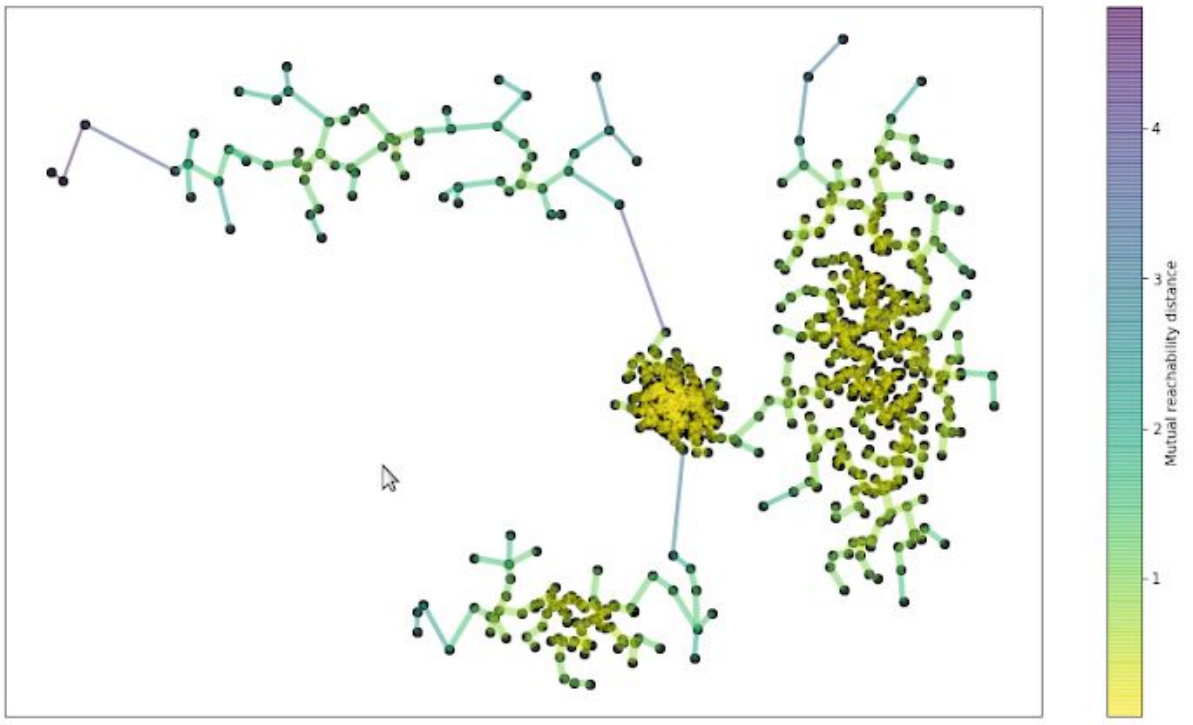


Comme on le voit dans le titre, `len(np.unique(hdbscan_d2c4.labels_))` = 5, **il nous trouve 5 clusters** dans ce dataset**.** Mais **…**

⚠ **Pas de conclusion hâtive :** En vérifiant comme pour DBSCAN les valeurs associées à chaque clusters, on se rend compte que le 5ème clusters à -1 comme valeur. Cette valeur étant attribuée au bruit nous avons donc bien **4 clusters !** Nous le verrons par la suite, dans les résultats, cette "classe" de bruitage n'est pas considérée comme un cluster à proprement parler.

Ainsi, sans toucher aux paramètres, HDBSCAN trouve le bon nombre de clusters. Il existe de nombreuses fonctions pour profiter d'un affichage de ces résultats ou d'avoir en graphique tout le processus d'arborisation qui mène à ces résultats.
Il faut pour cela ajouter des paramètres à notre modèle comme :

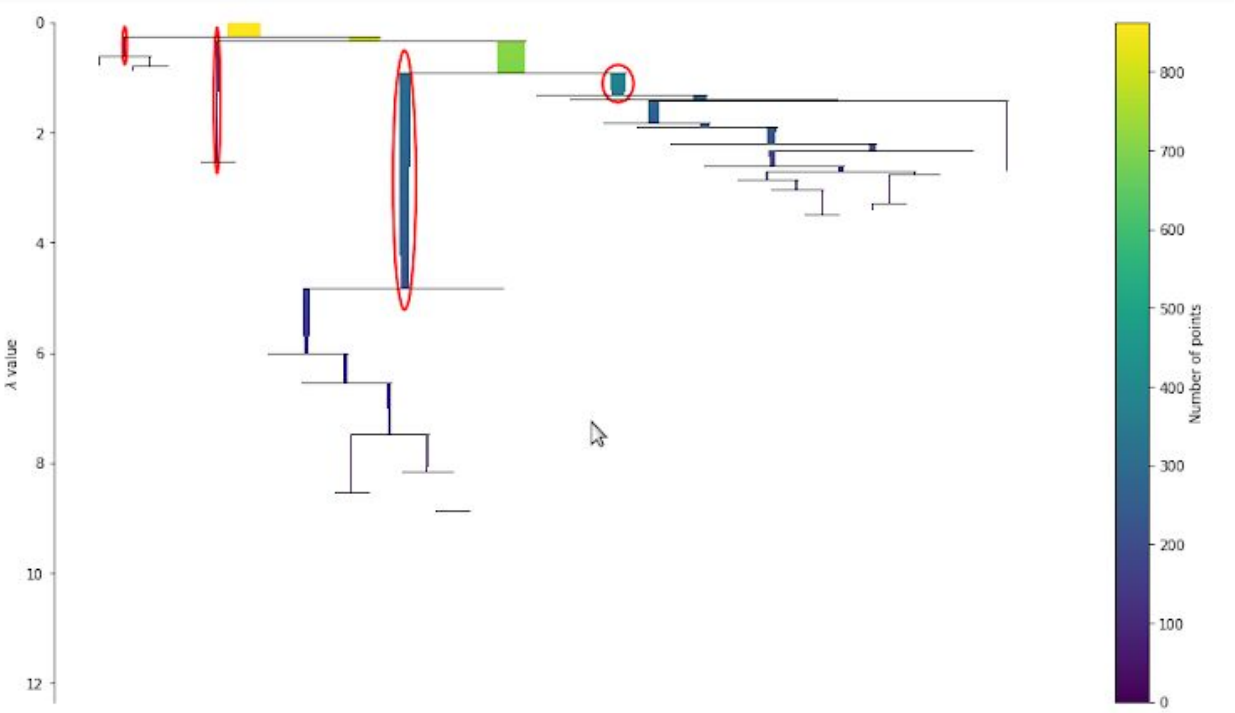
- **gen_min_span_tree = True**, pour avoir l'arbre ci-dessous.



Le MST (minimum spanning tree) pour le dataset D2C4

Ce graphe nous permet de voir efficacement les distances entre les points de notre dataset. Plus la couleur des arêtes est foncée et moins les points ont de chance d'être dans la même classe. On voit vite qu'au sein d'un même cluster, toutes les arêtes sont jaune ou bleu/vert clair. Ces couleurs sont compréhensibles avec l'échelle à droite qui montre la couleur en fonction de la distance entre les points.

Encore plus intéressant est le processus de hiérarchisation.



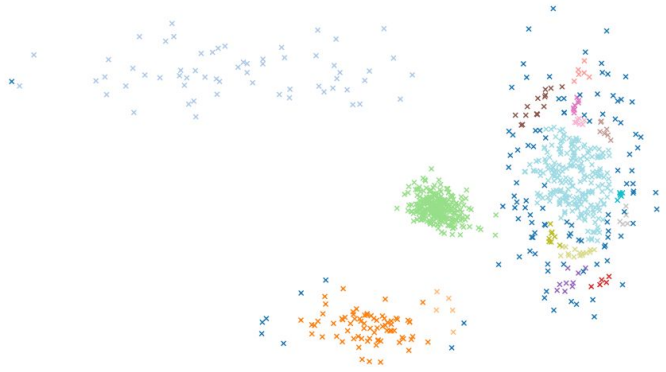
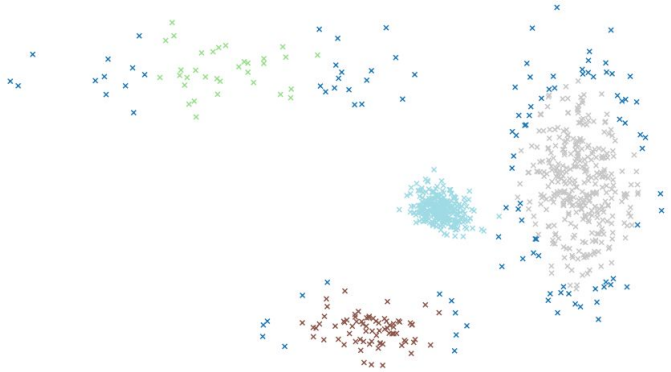
Hiérarchie des clusters pour D2C4

Ici, on voit quels branchements fictifs ont orienté le parcours des points du dataset, puis sur ces derniers on peut alors récupérer les différents clusters. Ici, on identifie facilement, entourés d'ovales rouges, les 4 clusters identifiés par le modèle.

Paramètres :

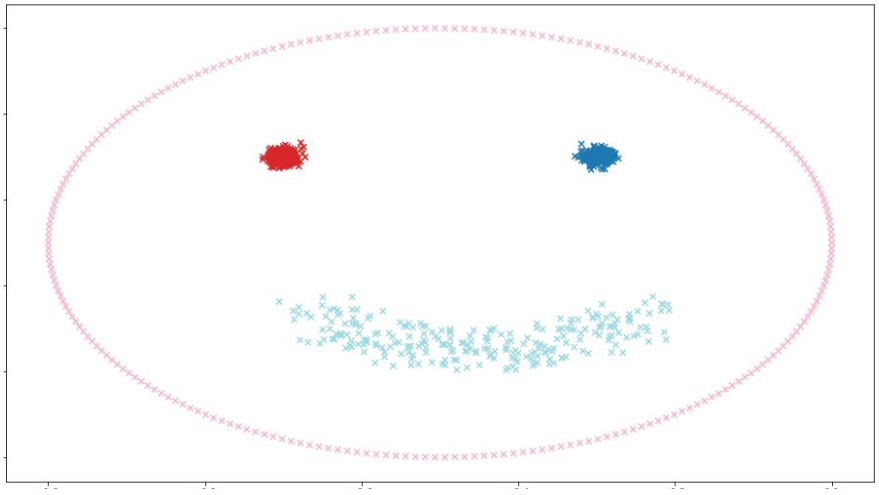
Nous avons testé de modifier plusieurs paramètres :

- p ⟶ aucun changement notable.
- min_samples ⟶ Nous remarquons qu'il est préférable de laisser la valeur de min_samples par défaut. Si on l'augmente, la part de bruit augmente aussi (ie. le nombre de points dans la classe bruitage augmente). Si on diminue cette fois la valeur de min_samples, on perds en précision et le nombre de clusters augmente très rapidement.

min_samples = 3 ⇒ 17 clusters, on est loin du compte !	**min_samples = 30** ⇒ 4 clusters mais beaucoup de bruit !
	

b) Dataset Smile

Passons maintenant au dataset avec des formes non-convexes et des densités bien définies. Sans toucher aux paramètres voici le résultat de HDBSCAN :



Incroyable ! Enfin ! Les résultats sont sans appel. Les 4 clusters sont parfaitement identifiés. Pas de bruit sur ce dataset et donc, aucun cluster de porte le label -1, nous avons donc 4 "vrais" clusters et c'est ce que nous attendions.

La différence est flagrante entre DBSCAN et HDBSCAN, l'un ne pouvait tout simplement pas gérer les formes non-convexes, alors que l'autre sélectionne parfaitement les clusters dans ce deuxième dataset.

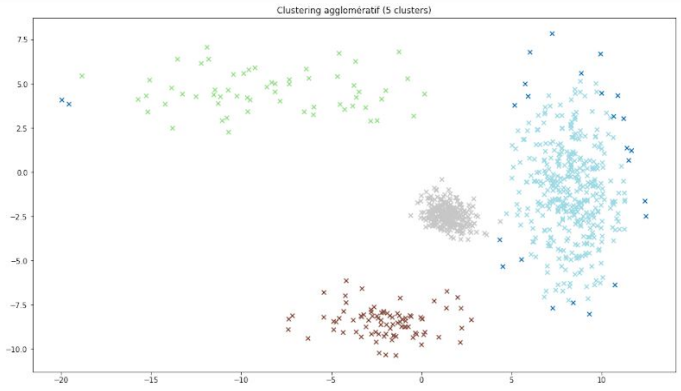
- ## Sensible à la nature des formes (cercle, rectangle, losange, non convexes, …)

La partie “b) Dataset Smile” démontre parfaitement la réponse à cette question. Les formes sont très bien prises en compte par HDBSCAN. Là où nous avions, en vain, essayé toutes les combinaisons possibles pour DBSCAN pour avoir un résultat un peu cohérent, ici en ne touchant quasiment rien, les 4 clusters sont identifiés sans bavures.
Cette particularité est un gros plus pour ce dernier algorithme.

- ## Sensibilité à la densité des données ?

HDBSCAN semble être sensible à la densité des données. En effet, ce problème se rapproche du bruit dans le sens ou si la densité d’un cluster est trop faible, il y a une chance non nulle que certains points soient considérés comme hors du cluster, donc comme du bruit.
On pourrait croire que les paramètres nous permettraient de passer outre cet inconvénient mais hélas non, car les densités de chaque clusters étant différentes, choisir un paramètre pour un cluster, diminue parfois la précision sur un autre. Peut-être que notre connaissance de l’algorithme n’est pas assez approfondie pour trouver les bonnes valeurs des paramètres en fonction du dataset.

- ## Le bruit dans les données est-il bien identifié ?



Le bruit est bien identifié. Pas aussi précisément qu’on pourrait le vouloir mais, comme le montre la première partie de cette section sur HDBSCAN, beaucoup mieux que dans DBSCAN son prédécesseur.

6. Synthèse

Dans cette dernière partie nous allons comparer les 4 méthodes de clustering détaillées plus haut dans ce rapport :

- clustering k-Means
- clustering agglomératif
- DBSCAN
- HDBSCAN

Les datasets *d2c4* et *smile* présentant des caractéristiques différentes des points de vue formes, convexité et densité (cf 1. Jeux de données), ils permettent d'identifier les principaux avantages et inconvénients de ces méthodes. Nous allons tout d'abord présenter les résultats obtenus lors de nos expérimentations dans un tableau, avant de les détailler.

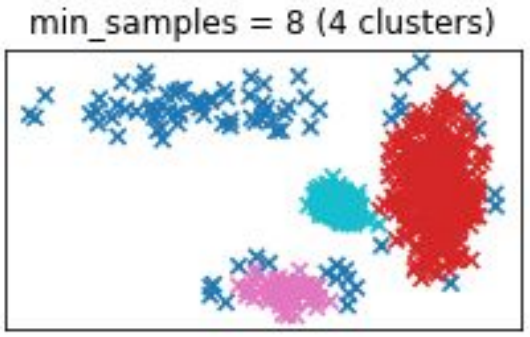
	Clustering k-Means	**Clustering agglomératif**	**DBSCAN**	**HDBSCAN**
Paramètres importants	*n_clusters* : nombre de clusters	*n_clusters* *linkage* : critère à minimiser pour la fusion de clusters *affinity* : distance	*min_samples* : nombre de points qand le voisinage d'un point pour qu'il ne soit pas considéré comme du bruit *eps* : distance maximale entre deux points qu'ils appartiennent à un même voisinage	*min_cluster_size* : taille minimale d'un regroupement de points pour qu'il soit considére comme un cluster *min_samples* *eps*
Détection de formes non-convexes	Non	Oui si *linkage=single*	Non	Oui
Sensibilité à la densité	Légèrement	Non	Oui	Légèrement
Sensibilité au bruit	Oui	Oui	Non	Non

Avec les clusterings k-Means et agglomératif, il faut passer le nombre de clusters souhaité en paramètre. Dans le cas de datasets plus complexes que ceux vus dans ce TP, notamment avec davantage de dimensions, on ne peut pas se faire une idée visuelle du nombre de clusters idéal. Il faut donc tester des nombres de clusters variés, ce qui prend du temps. Une solution, comme nous en avons discuté précédemment, serait d'ajouter une heuristique de recherche, mais ce n'est pas inclus de base dans les méthodes k-Means et de clustering agglomératif.

De plus ces deux méthodes **(k-Means et agglomérative) sont très sensibles au bruit**. En effet, chaque point est attribué à un cluster, au contraire de DBSCAN et HDBSCAN qui sont capables de créer un

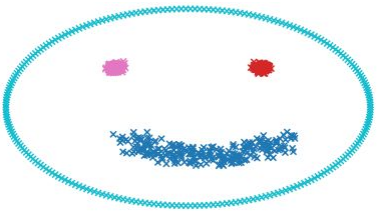
regroupement de points considérés comme du bruit. La détection du bruit de DBSCAN et HDBSCAN est bien sûre conditionnée par le paramétrage de la méthode utilisée.

Cependant nous avons pu voir avec DBSCAN, sur le dataset *d2c4*, que **DBSCAN est sensible à la densité**. Il a ainsi tendance à considérer les clusters peu denses comme du bruit (en bleu sur le graphe ci-contre).



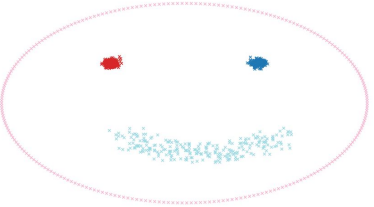
clustering k-Means

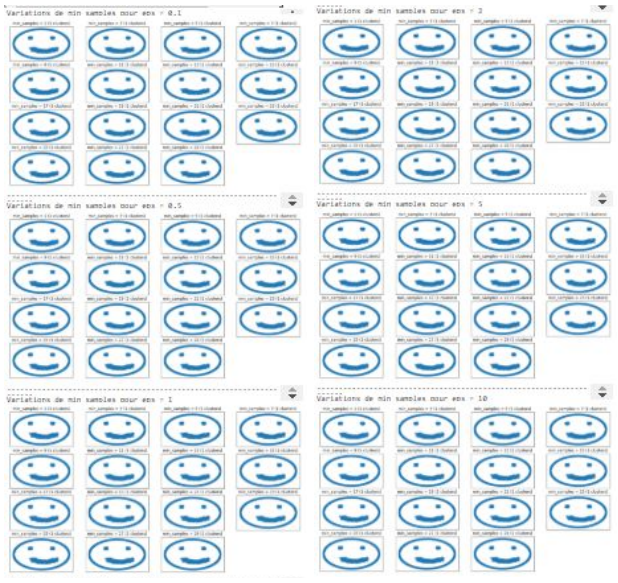
clustering agglomératif



Pour les formes non-convexes, comme sur le dataset *smile*, toutes les méthodes ne permettent pas de les identifier. **Le clustering k-Means n'est pas adapté aux formes non-convexes au contraire du clustering agglomératif**.

Avec un HDBSCAN basique (sans jouer avec les paramètres) sur le dataset *smile*, tous les clusters non convexes sont identifiés et cela malgré les différences de densité. **HDBSCAN gère bien les formes non convexes et est plutôt facile à paramétrer.**





Pour la méthode DBSCAN, nous n'avons pas réussi à distinguer les clusters non convexes. Cela provient sûrement du paramétrage du modèle. Cependant, bien qu'ayant testé de nombreuses combinaisons de *min_samples* et *eps*, nous n'avons pas réussi à obtenir un résultat concluant. **L'un des inconvénients de DBSCAN est donc la difficulté à le paramétrer.**

Pour conclure, on peut identifier les principaux avantages et inconvénients de chaque méthode.

	Avantages	**Inconvénients**
Clustering k-Means	* Peu de paramètres à fixer * Peu sensible à la densité	* Ne détecte pas les formes non convexes * Sensible au bruit
Clustering agglomératif	* Peu de paramètres à fixer * Détecte les formes non convexes * Robuste aux densités variées	* Sensible au bruit
DBSCAN	* Robuste au bruit	* Difficile à paramétrer * Ne détecte pas les formes non convexes * Sensible aux densités variées
HDBSCAN	* Détecte les formes convexes * Robuste aux densités variées * Robuste au bruit * Plutôt facile à paramétrer	

Finalement en regardant ce dernier tableau, la méthode qui semble la plus universelle est HDBSCAN, son seul inconvénient pouvant être son paramétrage, qui est tout de même bien plus facile que celui de DBSCAN.