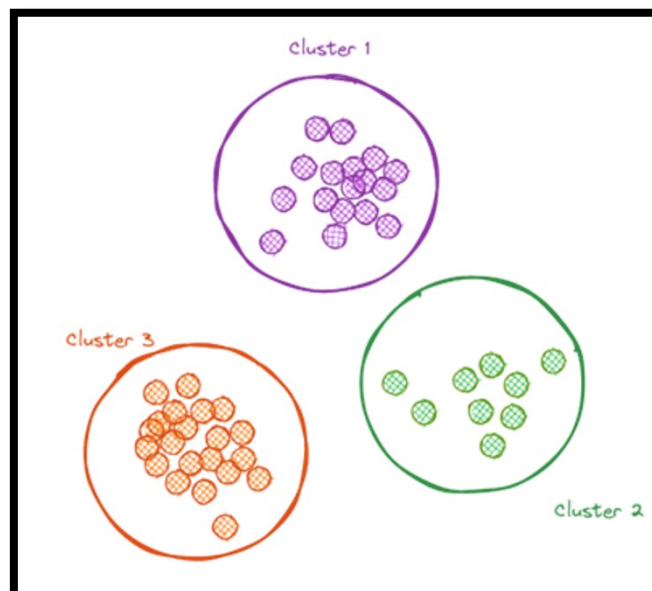

Rapport de TP Clustering



Introduction

Le cours d'Apprentissage nous a offert l'occasion de mettre en œuvre et de comparer différents algorithmes de clustering. Les jeux de données utilisés sont artificiels et en 2 dimensions pour des raisons pédagogiques.

Nous allons tout d'abord tester et analyser les différentes méthodes de clustering sur un large panel de jeux de données tout en faisant varier certains facteurs. Grâce à cela, nous allons pouvoir déterminer les intérêts et limites de chaque méthodes.

Lien de notre dépôt Git : <https://github.com/iSpeX1/clustering/>

1 Clustering K-means

Prise en main

K-means est une méthode populaire se basant sur un nombre K de clusters au préalable fixé pour déterminer les centres de gravité permettant de séparer les données. Chaque point sera attribué au cluster appartenant au centre de gravité le plus proche.

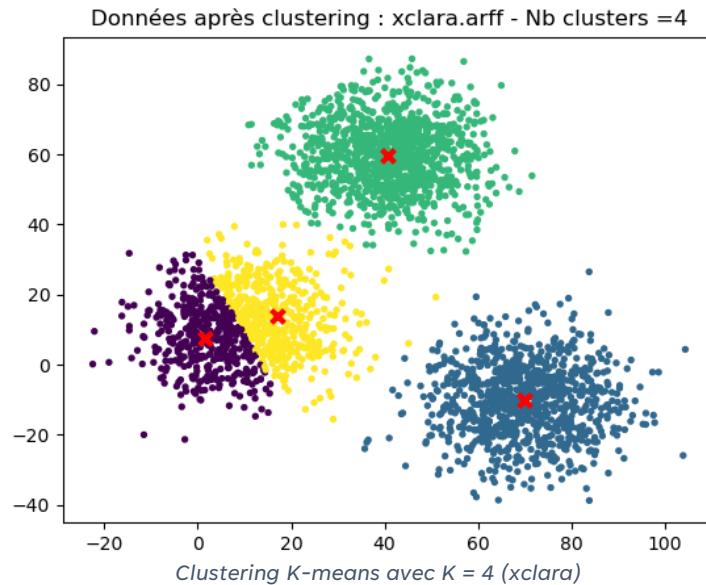
Nous avons commencé par calculer les scores de **regroupement** et de **séparation** du jeux de données *xclara.arff* avec la valeur arbitraire de **K = 4**.

Regroupement :

Distance Cluster	Min	Max	Moyenne
1	0,485	30,371	9,837
2	0,945	39,182	12,997
3	0,380	32,194	12,354
4	0,203	34,297	10,641

Séparation :

Min	Max	Moyenne
16,876	75,866	56,430

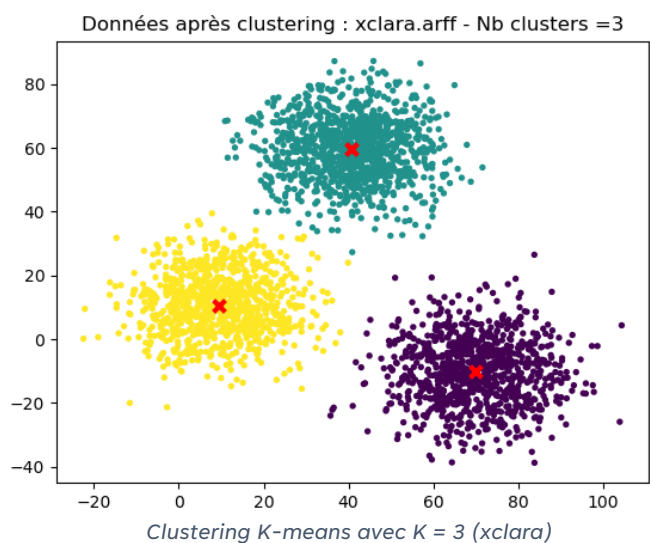
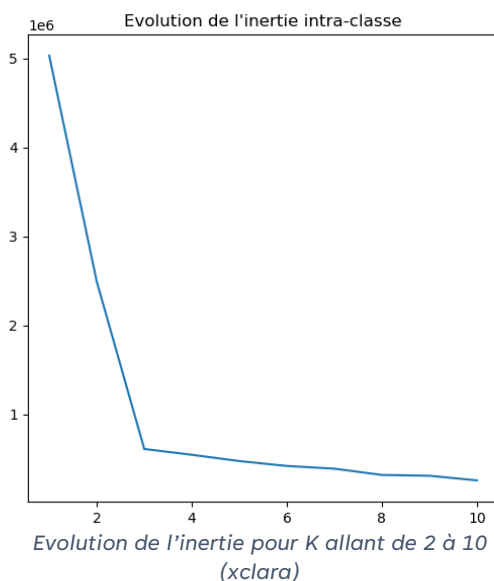


Nous pouvons observer ci-dessus que la répartition des données n'est pas optimale avec la valeur $K=4$. C'est pourquoi il existe différentes méthodes afin de calculer le bon nombre de cluster à utiliser.

Application itérative basée sur l'inertie

Afin de déterminer le **nombre optimal de cluster** pour un jeu de donnée, nous pouvons utiliser une méthode graphique basée sur l'inertie : *la méthode du coude*. En effet, en faisant varier itérativement la valeur de K, nous pouvons observer une **évolution de l'inertie**. Le nombre de cluster optimal est alors **déductible graphiquement** en considérant le **point d'inflexion**.

Nous remarquons ici que le point d'inflexion est situé à $K = 3$.



Les scores de **regroupement** et de **séparation** de la solution obtenus sont :

Regroupement :

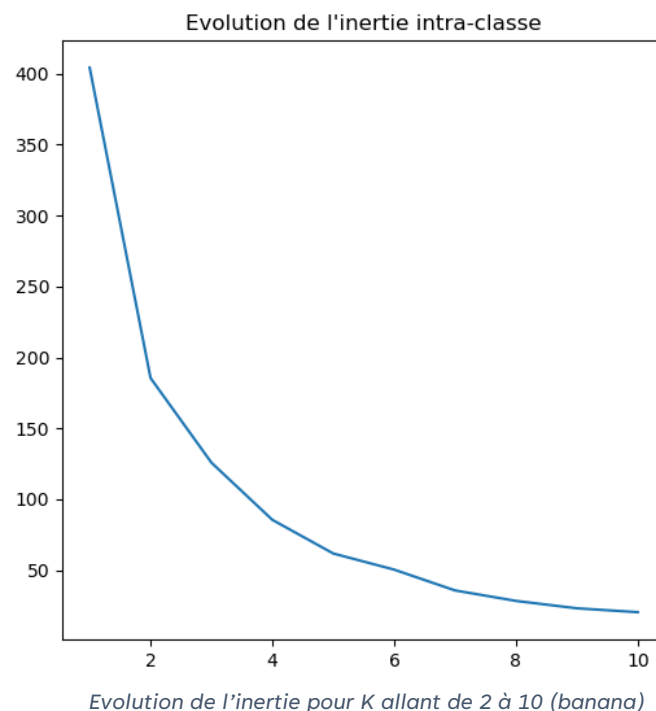
Distance Cluster	Min	Max	Moyenne
1	0,290	37,212	12,580
2	0,941	39,154	13,037
3	0,443	32,400	12,439

Séparation :

Min	Max	Moyenne
58,118	75,710	65,918

La **méthode du coude** n'est pas très optimale car cela ne fonctionne que sur les jeux de données où l'évolution de l'inertie possède un point d'inflexion facilement identifiable. Cela n'est pas toujours le cas et cette méthode ne permet pas de ressortir le K optimal automatiquement au sein du code.

Exemple : *banana.arff*



Application itérative basée sur le coefficient de silhouette

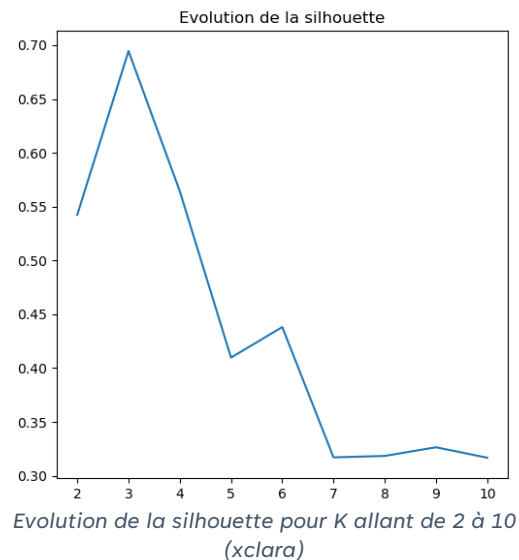
Afin d'évaluer le résultat d'un clustering, il existe d'autres métriques : coefficient de silhouette, indice de Davies-Bouldin et indice de Calinski-Harabasz.

Dans la suite du TP, nous avons décidé d'utiliser le **coefficient de silhouette** afin de déterminer le bon nombre de cluster à utiliser au sein de chaque jeux de données.

Exemple : *xclara.arff*

K	2	3	4	5	6	7	8	9	10
Silhouette	0,542	0,695	0,564	0,410	0,438	0,317	0,318	0,326	0,316

Temps de calcul total : 0,67s



Afin de déterminer le nombre optimale de cluster, il nous suffit de regarder la **valeur la plus élevée** de la silhouette et de sélectionner le **K associé**.

Comme déterminé avec la méthode du coude, ici le **K optimal est égal à 3**.

Les **scores de regroupement** et **de séparation** de la solution obtenus sont alors **identiques** à ceux calculés précédemment, la valeur de K étant identique :

Regroupement :

Distance Cluster	Min	Max	Moyenne
1	0,290	37,212	12,580
2	0,941	39,154	13,037
3	0,443	32,400	12,439

Séparation :

Min	Max	Moyenne
58,118	75,710	65,918

Intérêts et Limites

Intérêts

L'intérêt principal de la méthode *K-means* est sa **facilité d'utilisation**. Cette méthode est relativement **simple à comprendre et à implémenter** et c'est sûrement pourquoi cette dernière est si populaire.

De plus, son **interprétabilité** est assez **intuitive** car chaque point est assigné à un **cluster unique**.

Limites

Initialement, la méthode *K-means* est **sensible à l'initialisation**¹ mais nous n'avons pas été soumis à cette limite grâce à l'initialisation *K-means ++* que propose scikit-learn sélectionnant le premier centre de manière aléatoire puis répartissant les suivants à l'aide d'une probabilité calculée.

De plus, cette méthode suppose que les **clusters sont de forme sphérique** et ont une **variance égale dans toutes les directions**. Elle n'est alors pas appropriée pour les jeux de données ayant des **formes complexes**.

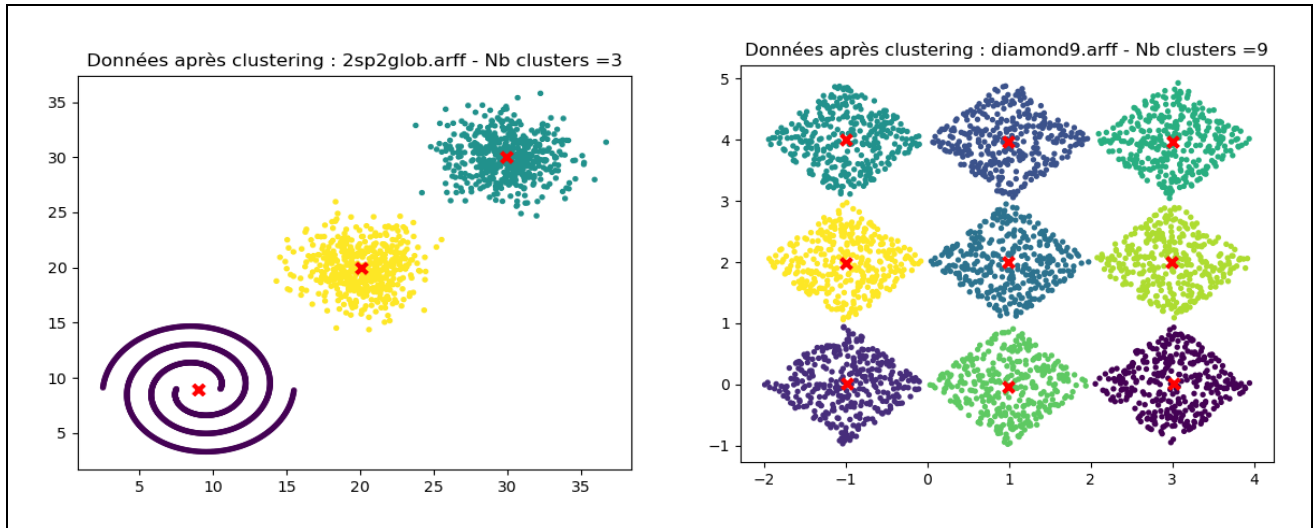
Pour finir, *K-means* nécessite de **spécifier le nombre de cluster désiré** et n'est alors pas efficace sans l'utilisation de méthodes alternatives calculant la valeur optimale de K telle que la **méthode du coude** ou le **calcul de la silhouette**.

¹ Dépendant de l'initialisation des centres des clusters, différentes initialisations pouvant conduire à différentes solutions.

Exemples d'exécutions

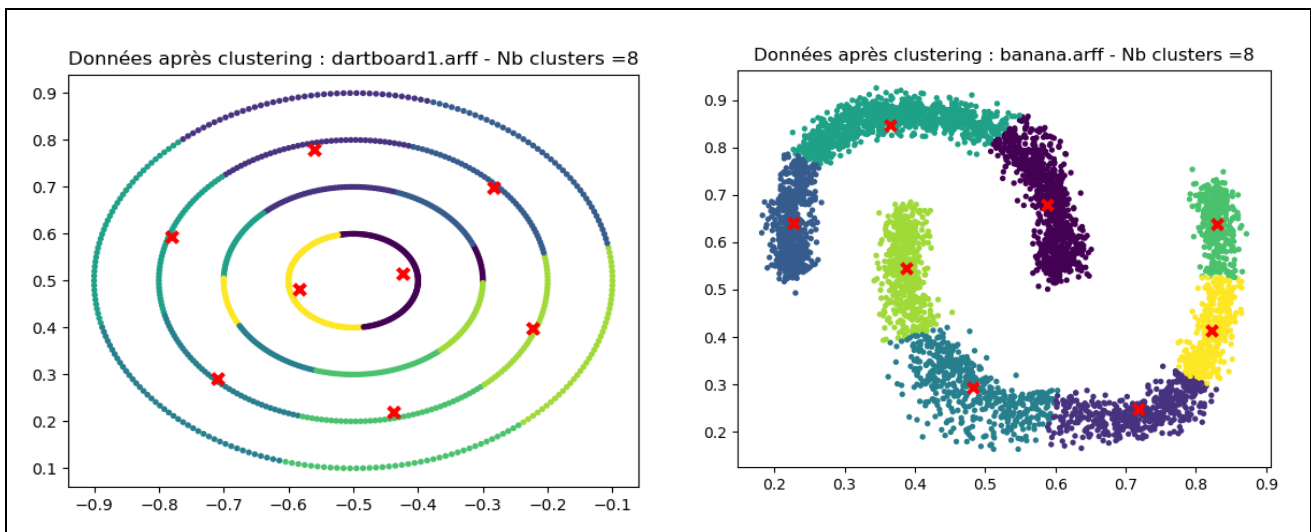
Bon clustering :

- Jeux de données avec des formes simples, espacées uniformément.



Mauvais clustering :

- Jeux de données avec des formes complexes.
- Clusters avec une variance non équivalentes dans chaque direction.

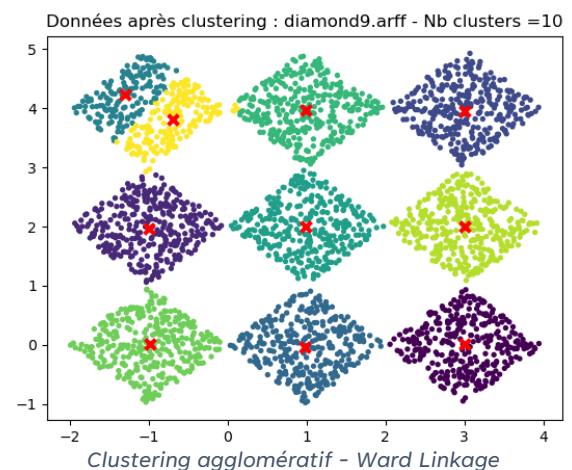
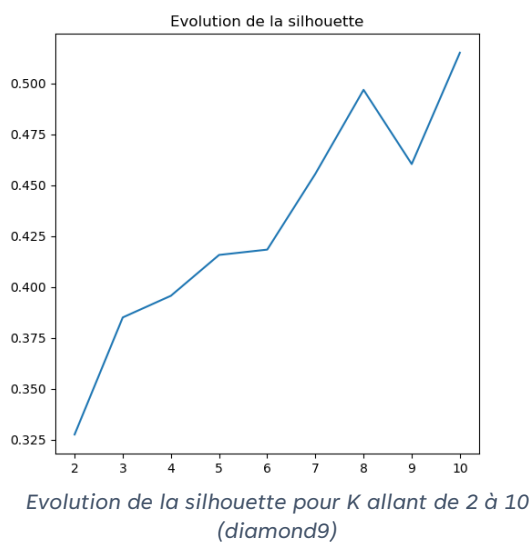
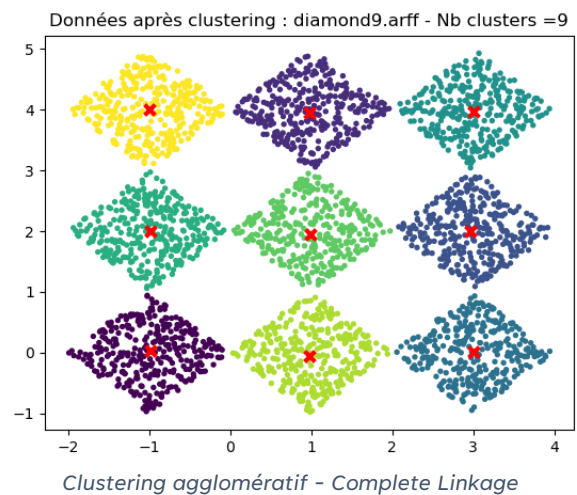
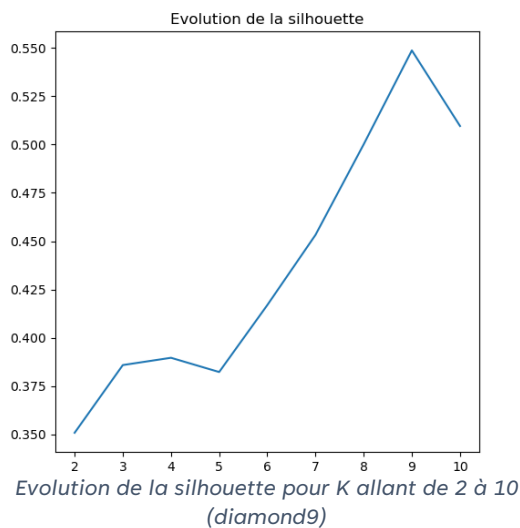


Version mini-batch

Le **mini-batch** est une variante de la méthode *K-means* utilisant des mini-batch de données plutôt que l'ensemble complet. En effet, au lieu de mettre à jour le centre des clusters après chaque point de données, cette méthode le fait périodiquement avec un **échantillon aléatoire de données**.

Cette approche accélère le processus, le rendant avantageux sur les grands jeux de données mais peut proposer un résultat légèrement **plus aléatoire** que la version standard.

Comme nous le voyons ci-dessous, pour deux exécutions identiques sur le jeu de données diamond9.arff, nous avons une bonne solution et une mauvaise.



Les jeux de données que nous avons testés n'étant pas assez grand, nous n'obtenons pas de différences significatives concernant le temps d'exécution.

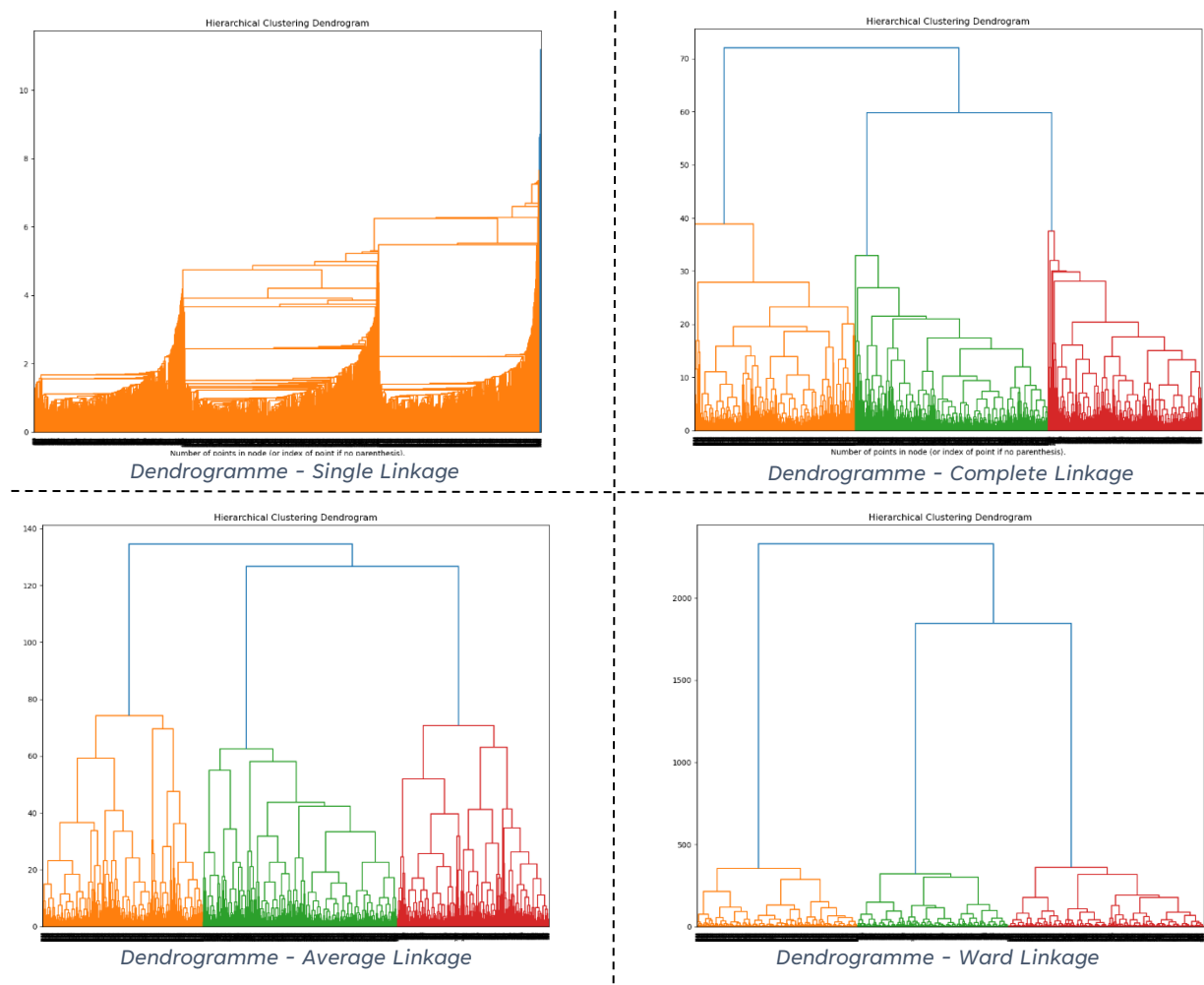
2 Clustering agglomératif

Prise en main

Le **clustering agglomératif** est une méthode de clustering **montante**² considérant à l'origine chaque point comme un cluster séparé. Le but est par la suite de **progressivement fusionner** les clusters optimaux afin de **créer une hiérarchie** de clusters emboîtés. Il existe plusieurs manières de combiner les clusters (**Linkage**) :

- **Single** : On utilise les plus petites distances entre deux clusters
- **Complete** : On utilise les plus grandes distances entre deux clusters
- **Average** : On utilise les distances moyenne entre deux clusters
- **Ward** : On minimise la variance intra-cluster lors de la fusion

Voici les **dendrogrammes**³ du jeu de données *xclara.arff* correspondants aux différentes méthodes de linkage possible au sein du clustering agglomératif.

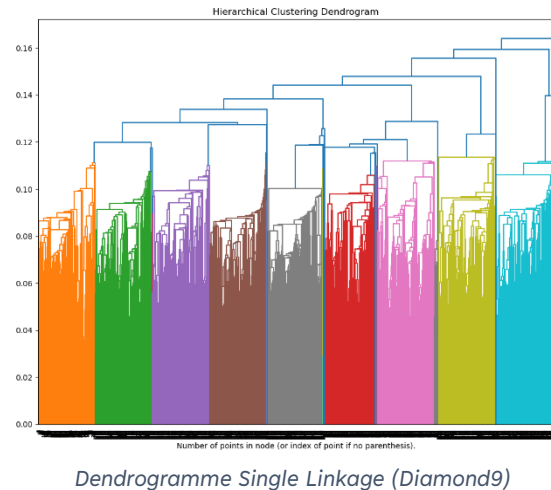
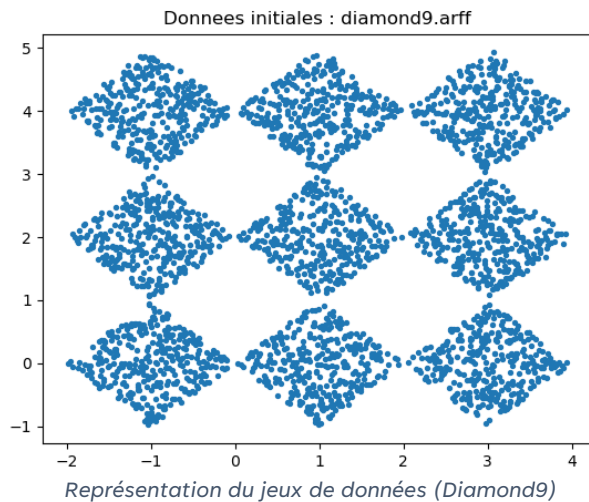


² Les méthodes descendantes, elles, considèrent initialement que tous les points appartiennent au même cluster.

³ Diagramme représentant la structure hiérarchique des clusters et leur fusion durant l'algorithme.

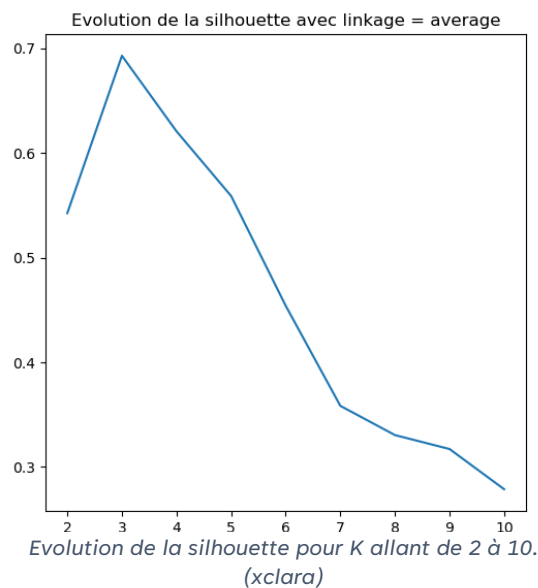
Nous pouvons voir ici les **différentes hiérarchies** générés par l'algorithme de clustering agglomératif et nous remarquons immédiatement une **similarité** entre les résultats des méthodes de linkage « *Complete* », « *Average* » et « *Ward* ».

La méthode « *Single* » étant **peu représentative** sur cet exemple, nous avons décidé de la tester sur d'autres jeux de données dont *Diamond9.arff* proposant un résultat un peu plus intéressant :



Application itérative – Linkage fixe

Nous avons appliqué itérativement la méthode **du clustering agglomératif** afin de déterminer le nombre de clusters obtenant le **meilleur coefficient de silhouette**. Pour cela, nous avons arbitrairement décidé d'utiliser le linkage « *Average* » sur le jeux de données *xclara.arff*.



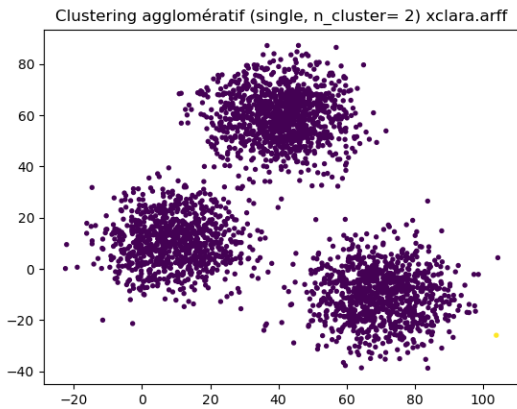
Nous nous apercevons ici que, comme pour *K-means*, le coefficient de silhouette est à son maximum pour **K = 3**. Tous les dendrogrammes exceptés « *Single* » terminaient bien l'exécution avec **3 clusters** et étaient alors **optimaux**.

Application itérative – Linkage variable

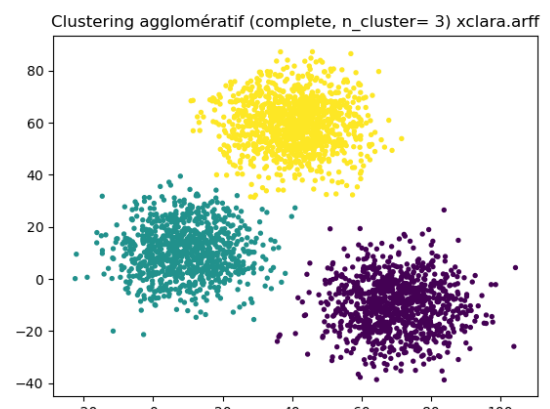
Ensuite, nous avons appliqué itérativement cette méthode pour **chaque méthode de linkage** afin de mesurer leur **temps de calcul en millisecondes**.

Linkage K	Single	Complete	Average	Ward
2	80.49	116.89	137.01	125.92
3	36.02	151.07	108.23	144.46
4	47.57	160.97	139.63	130.05
5	37.09	153.76	135.28	136.23
6	44.53	126.68	145.38	130.84
7	60.85	120.89	123.63	122.27
8	44.58	118.87	108.16	125.9
9	57.94	128.46	146.7	121.92
10	59.55	121.47	146 .01	121.64
Total	468,62	1199,06	1190,03	1159,23

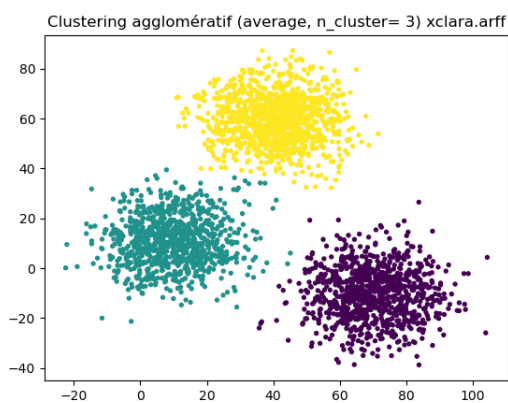
Nous remarquons encore une fois ici une **singularité** en terme de temps de calcul pour la méthode « *Single* » qui, itérativement, dure prêt de trois fois moins longtemps que les autres. Cela peut paraître avantageux mais impacte la véracité du clustering comme nous pouvons le voir ci-dessous :



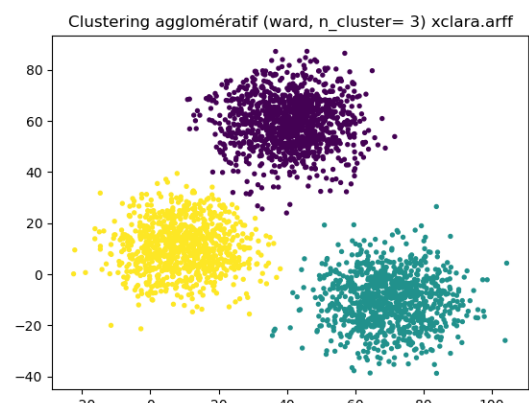
Clustering agglomératif - Single Linkage



Clustering agglomératif - Complete Linkage



Clustering agglomératif - Average Linkage



Clustering agglomératif - Ward Linkage

Intérêts et Limites

Intérêts

L'utilisation du clustering agglomératif a plusieurs intérêts. En effet, **son nombre de clusters n'est pas forcément fixé par l'utilisateur** mais peut être établi depuis le dendrogramme.

La **flexibilité** de la méthode de fusion est également un élément très important dans cette méthode. Grâce à cela, nous pouvons **adapter les critères de fusion** afin de **correspondre au maximum aux caractéristiques** des données sélectionnées.

Son **interprétabilité** est également facilitée grâce aux **dendrogrammes** permettant de **comprendre au mieux les relations** entre les clusters à différents niveaux de la hiérarchie.

Limites

Le clustering agglomératif est une méthode possédant des intérêts notables mais cette dernière comporte également des **limites**.

La première limite de cette méthode est sa **complexité**, en effet, étant donné que ses critères de fusion se font par rapport au calcul de distance, **elle ne conviendra pas aux larges jeux de données**.

De plus, le fait d'avoir **différentes méthodes de linkage** est une force mais aussi une faiblesse : le choix du critère de fusion peut avoir un réel impact sur les résultats (comme vu précédemment avec « *Single* »), il n'y a **pas de méthode unique** universellement meilleure.

Exemples d'exécutions

Clustering non optimal :

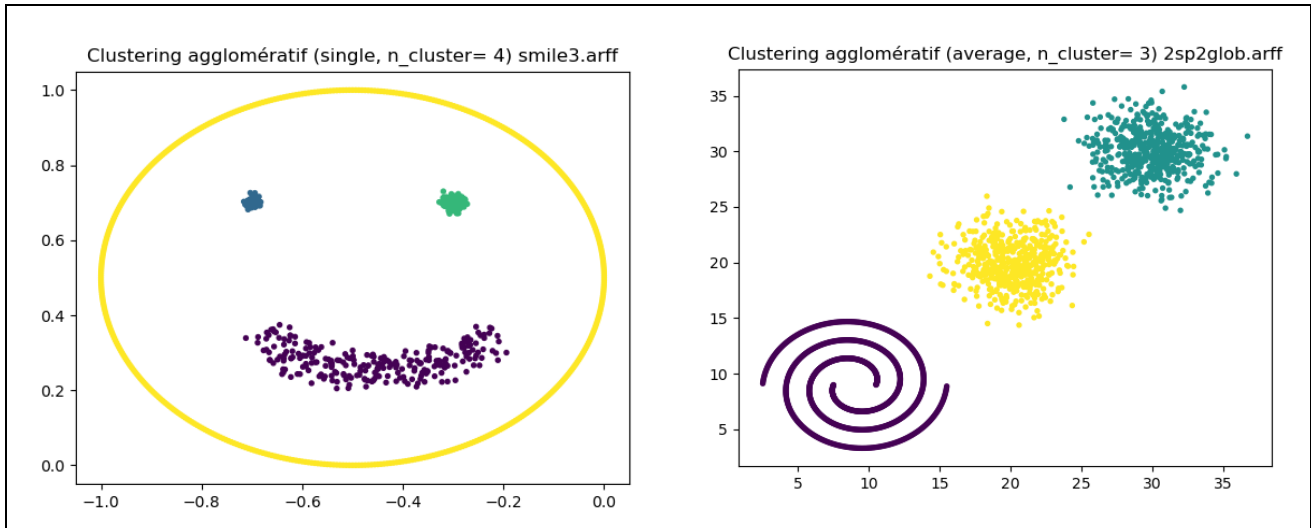
- Large jeux de données | **Exécution beaucoup trop longue**

Mopsi-finland.arff (13466 points) ~ 34 sec

Cluster	2	3	4	5	6	7	8	9	10	Total
Temps (s)	3,891	3,722	3,711	3,789	3,705	3,733	3,701	3,728	3,721	33,70

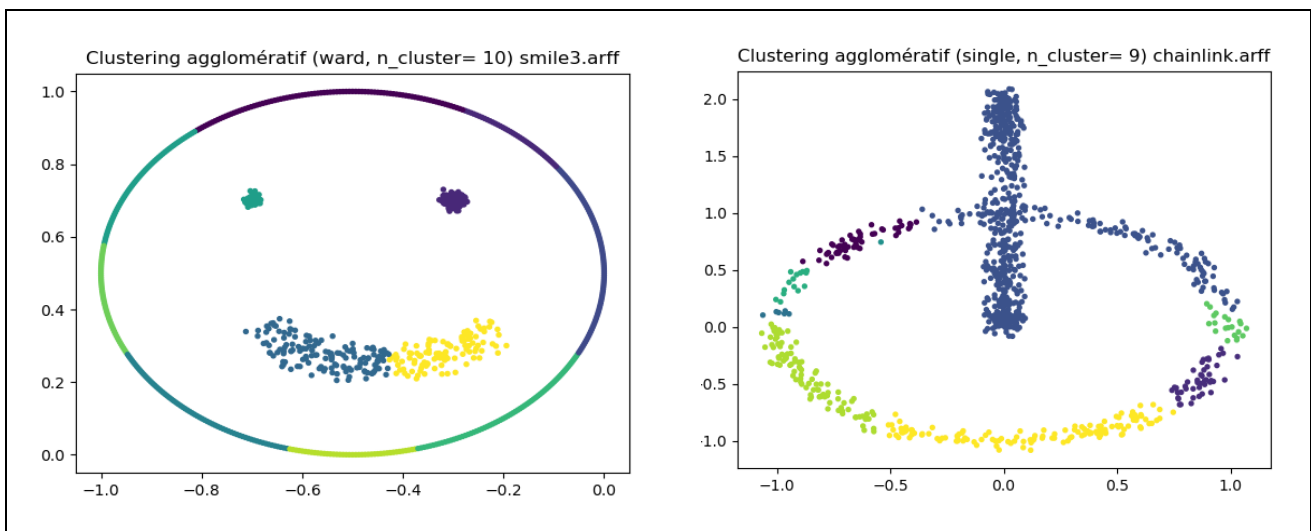
Bon clustering :

- Jeux de données avec des formes plus complexes, avec des espacements assez importants entre les différents clusters (Linkage Single)
- Jeux de données avec des formes simples, espacées uniformément. (Linkage Complete, Average & Ward) | **Peu de différence avec *K-means*.**



Mauvais clustering :

- Jeux de données avec des formes complexes mais mauvais choix de critère de fusion | **Ne fonctionne qu'avec Linkage Single**
- Jeux de données avec des formes complexes et peu d'espaces entre les différents clusters.



3 Clustering DBSCAN et HDBSCAN

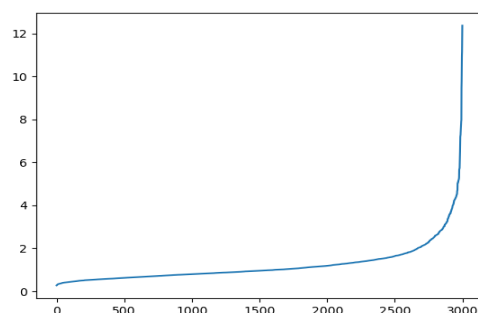
Prise en main

Le concept de cette méthode d'analyse repose sur **l'exploration du voisinage**. Nous allons visiter chaque point afin d'explorer ses points adjacents et ainsi déterminer s'il fait partie d'un voisinage. Dans un deuxième temps, si le nombre de voisins est assez élevé nous allons pouvoir considérer ce voisinage comme un cluster. Il y a donc deux paramètres à préciser pour utiliser cette méthode : la distance **epsilon** qui va permettre de déterminer le voisinage à partir d'un point de départ, et le **nombre minimum de point** présent dans un voisinage pour être considéré comme une partie du cluster. Autrement, ces points seront considérés comme du bruit.

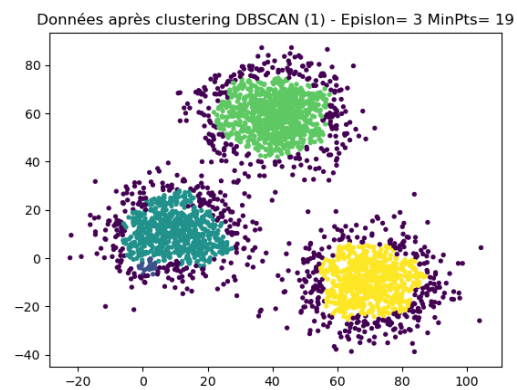
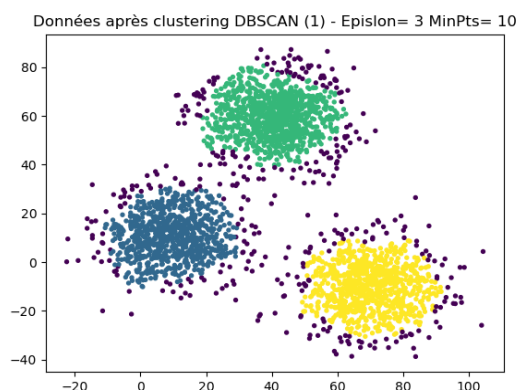
Application de la méthode DBSCAN

Des deux paramètres cités précédemment, le plus important des deux semble donc être Epsilon : celui-ci va permettre en grande majorité de déterminer le nombre de clusters d'un jeu de données. Afin de le paramétrer correctement nous utilisons donc une méthode citée précédemment : la **méthode du coude**. En analysant la distance moyenne des 5 plus proches voisins de chaque point d'un jeu de donnée, nous pouvons ainsi tracer une courbe en classant ces valeurs par ordre croissant. On détermine le "coude" de cette fonction et on obtient une valeur de Epsilon qui semble **optimale**.

Reprenons par exemple le jeu de données précédent *xclara*. Nous obtenons alors la courbe suivante.

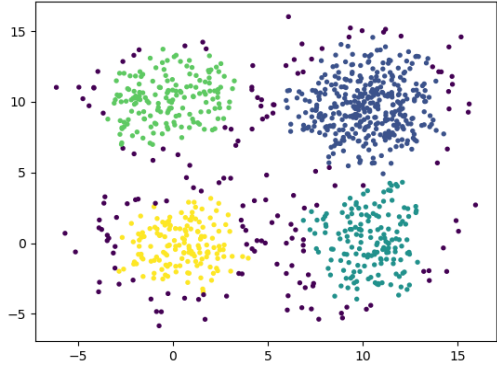


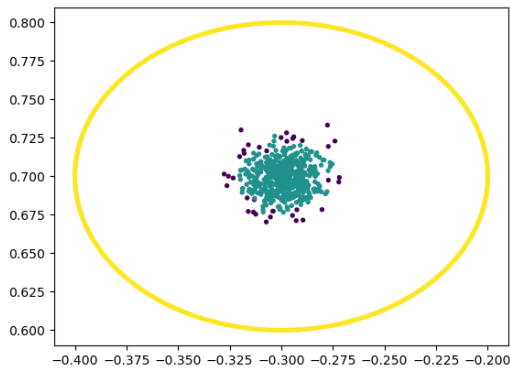
On distingue bien un coude à cette fonction autour d'une distance de 3. En paramétrant DBSCAN avec **epsilon = 3**, on obtient ainsi une analyse distinguant correctement les trois clusters. En agissant sur le paramètre MinPts, on constate bien que la quantité de points dans le cluster change, et donc que **le bruit augmente ou diminue** comme nous pouvons l'observer ci-dessous.

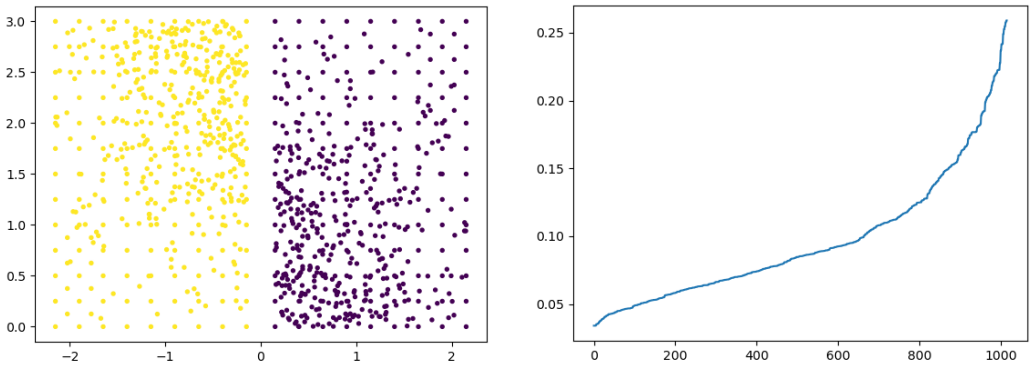


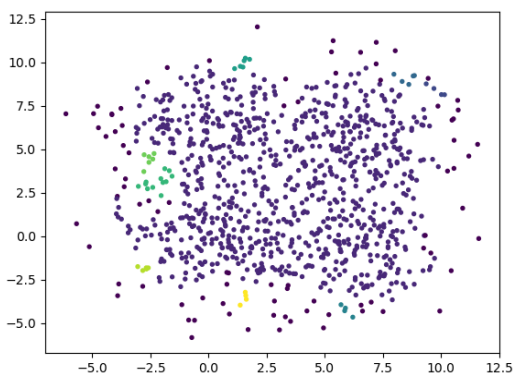
Intérêts et Limites – DBSCAN

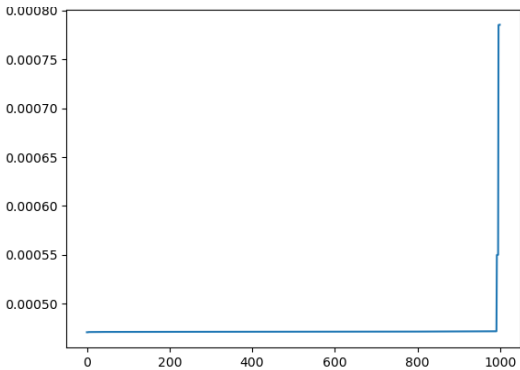
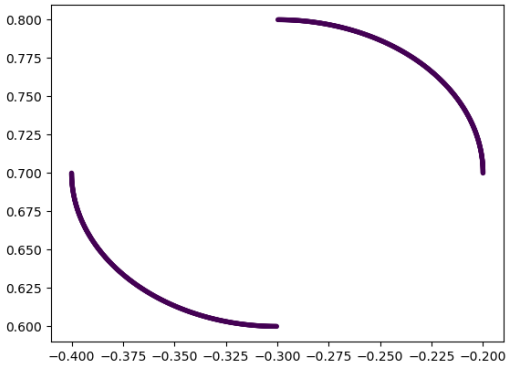
Afin de déterminer les Intérêts et inconvénients de cette méthode nous allons sélectionner plusieurs jeux de données et constater si cette méthode peut être réellement efficace. Nous exprimerons d'abord une intuition sur la manière dont ce jeu de données sera analysé puis nous pourrons ainsi la confirmer ou l'infirmer selon les résultats obtenus.

Dataset	sizes1.arff
Intuition	A première vue ce dataset semble facile à analyser avec DBSCAN. Les clusters se distinguent bien visuellement et la méthode permettrait de discerner le bruit.
Illustration	 <p>DBSCAN avec $Epsilon=1$ et $MinPts=10$</p>
Vérification	Dans les fait la méthode permet donc bien de déterminer les 4 clusters, comme sur l'image ci-dessus. Mais le paramétrage est assez compliqué, même avec la méthode du coude. Sans analyse mathématique de la courbe, la valeur de epsilon se détermine parfois par dichotomie jusqu'à avoir un résultat convenable.

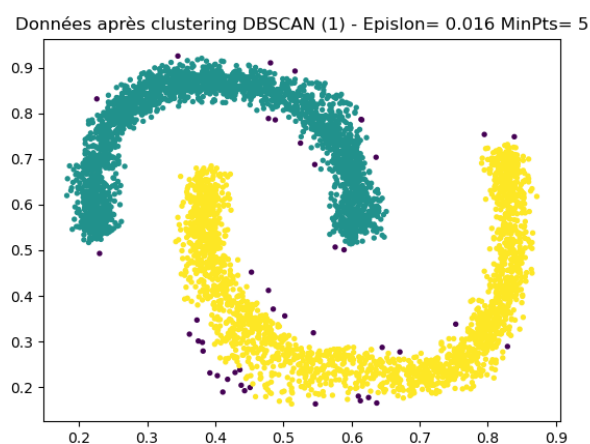
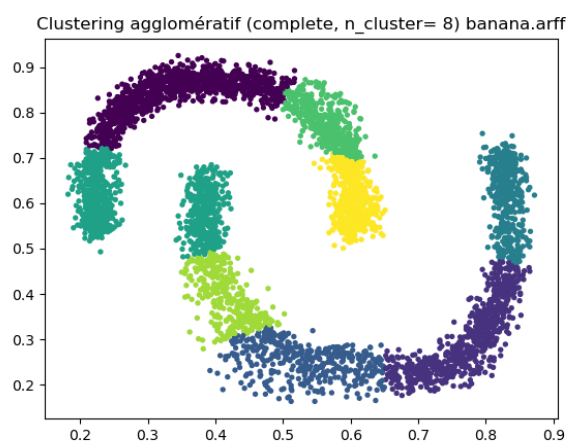
Dataset	donut1.arff
Intuition	Globalement la même intuition que pour sizes1, les clusters sont espacés donc ne devraient pas être trop complexe à analyser
Illustration	 <p>DBSCAN avec $Epsilon=0.0035$ et $MinPts=4$</p>
Vérification	Les clusters sont bien analysé et différenciés à condition que l'algorithme soit bien paramétré encore une fois; on note tout de même des difficulté au niveau du bruit, le cercle extérieur est correctement groupé mais quel que soit le paramétrage, une partie du cercle intérieur sera quasiment toujours considéré comme du bruit

Dataset	wingnut.arff
Intuition	Les deux cluster devraient être correctement identifiés étant donné la séparation entre les deux nuages de points
Illustration	 <p><i>DBSCAN avec Epsilon=0.25 et MinPts=5</i></p> <p><i>Moyenne de distances aux 5 plus proches voisins</i></p>
Vérification	En effet, une fois bien paramétré on identifie 2 clusters distincts et pas de bruit. Cependant pour certaines courbes, comme celle-ci, le "coude" peut être difficile à déterminer. Il n'est pas évident de voir que la valeur optimale de epsilon est 0.25.

Dataset	square5.arff
Intuition	Ce jeu de données semble difficile à analyser car très bruité, il apparaîtrait que 3 clusters doivent être identifiés mais ils se superposent
Illustration	 <p><i>DBSCAN avec Epsilon=0.6 et MinPts=4</i></p>
Vérification	En effet les clusters distingués ne sont pas du tout ceux attendus, selon le paramétrage nous auront ou bien un grand cluster principal, ou bien de petits clusters au milieu de beaucoup de bruit. Mais jamais les 3 grandes zones attendues

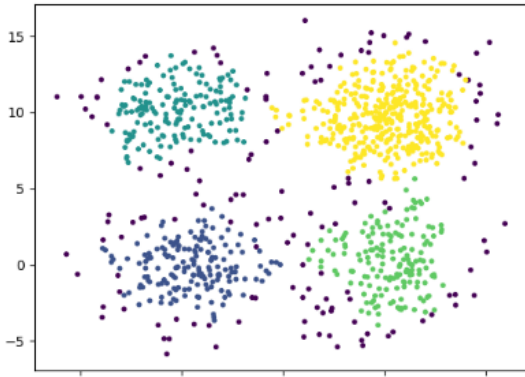
Dataset	curves1.arff	
Intuition	Ici, les clusters semblent faciles à discerner mais les points étant très espacés, le coude sera probablement très abrupt	
Illustration	 <p>Moyenne de distances aux 5 plus proches voisins</p>	 <p>Jeu de donnée curves1, aucun cluster discerné</p>
Vérification	En effet on observe quasiment une courbe en angle droit, ce qui rend très compliqué d'estimer une valeur de epsilon et donc de paramétrer correctement l'algorithme. Je n'ai donc pas pu trouver de valeur pour distinguer les deux clusters	

DBSCAN présente donc un certain nombre d'avantages. On peut citer notamment le paramétrage du nombre de clusters qui est absent. Le fait que DBSCAN permet de déterminer le nombre de clusters par elle-même rend cette méthode très pratique et **beaucoup moins rigide** que les méthodes mentionnées précédemment. La **détection du bruit** apporte également un énorme intérêt qui n'était pas possible précédemment. Cet algorithme permet également de détecter des clusters de **formes convexes**.

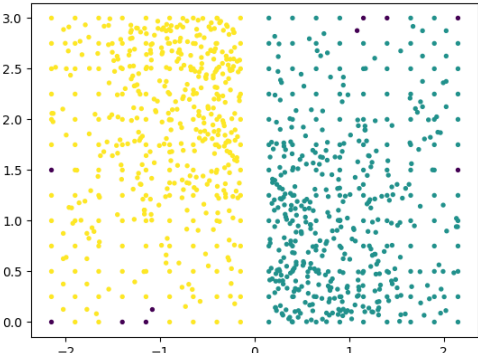


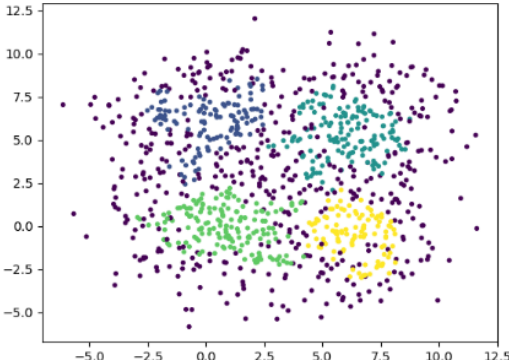
Comparaison avec HDBSCAN

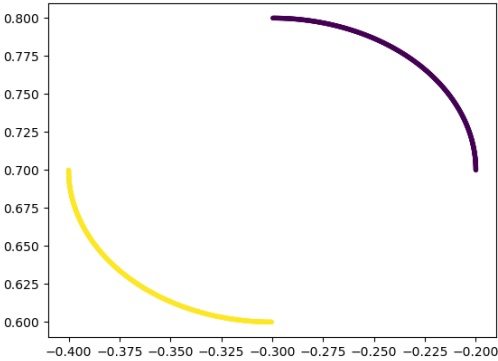
Nous utilisons maintenant l'algorithme d'analyse de cluster **HDBSCAN**, une extension de l'algorithme précédent. Nous allons donc pouvoir **comparer** leur efficacité en se basant sur les exemples précédemment mentionnés. L'algorithme se configure grâce à deux paramètres : ***min_cluster_sizes*** et ***min_samples***. Le premier permet de régler la taille minimale des clusters que l'algorithme doit trouver, et le second permet de régler le nombre minimum de point pour considérer un cluster.

Dataset	size1.arff
Comparaison	On obtient une bonne analyse des clusters même ayant <i>min_cluster_sizes</i> variant entre 10 et 100. Quel que soit la valeur de ce paramètre l'analyse sera toujours la même.
Illustration	 <p>HDBSCAN avec <i>min_cluster_sizes</i> de 10 à 100 et <i>min_pts</i>=8</p>

Dataset	donut1.arff
Comparaison	L'analyse est la même que pour DBSCAN, cependant le centre du cercle représente un cluster entier et il n'y a pas de bruit .

Dataset	wingnut.arff
Comparaison	Dataset légèrement moins bien analysé que précédemment, un peu de bruit est détecté dans les deux zones. Le problème peut cependant venir de notre paramétrage.
Illustration	 <p>HDBSCAN avec <i>min_cluster_sizes</i>=2 et <i>min_pts</i>=10</p>

Dataset	square5.arff
Comparaison	Dataset très long à paramétrer avant d'avoir un résultat convenable. Les différents clusters analysés ressemblent déjà plus à ceux attendus qu'avec DBSCAN mais ne sont cependant toujours pas correct. Cela révèle un problème de méthode : nous ne disposons pas ici de courbe permettant de choisir un paramétrage convenable.
Illustration	 <p>HDBSCAN avec $min_cluster_sizes=25$ et $min_pts=8$</p>

Dataset	curves1.arff
Comparaison	Contrairement à DBSCAN les deux clusters ont été détecté directement et sont correctement analysé pour une grande échelle de valeur de $min_cluster_sizes$
Illustration	 <p>HDBSCAN avec $min_cluster_sizes=100$ et $min_pts=4$</p>

Ne disposant pas de méthode pour trouver les bons paramètres comme précédemment avec la méthode du coude, la configuration de cette version peut être relativement longue pour certains jeux de données. Cependant dans la majorité des cas la configuration optimale se trouve rapidement. Le paramètre $min_cluster_sizes$ étant le plus important dans la détermination des clusters, la marche à suivre est celle-ci : si l'on a une idée approximative de la taille du plus petit cluster de notre jeu de donnée, on peut régler le paramètre sur cette valeur. Autrement il est préférable de commencer avec une petite valeur et d'incrémenter progressivement jusqu'à atteindre le résultat voulu. Dans la majorité des cas, une fois la détermination des bons clusters faite, augmenter la valeur de ce paramètre (dans une certaine mesure) ne changera plus le résultat et les clusters resteront les mêmes.

Conclusion

Cette étude des différentes méthodes de clustering nous a offert l'opportunité d'illustrer et de mettre en pratique les concepts abordés en cours tout en nous familiarisant avec la librairie Scikit-learn. En abordant des jeux de données variés en terme de taille et de structure, nous avons pu approfondir notre compréhension des forces et des limites de chaque méthode.

La liberté d'autonomie dont nous avons bénéficié pour découvrir ces méthodes et explorer les différents jeux de données nous ont parfois causée quelques difficultés mais cela nous a également permis de se détacher du cadre académique et de développer une approche curieuse et critique du problème.

Ce projet en Python représente à notre goût une approche très intéressante de l'apprentissage et de l'intelligence artificielle, à travers une approche concrète et stimulante.

Voici un tableau récapitulatif des différentes méthodes observées au sein de ce TP et leurs caractéristiques :

Méthode	Caractéristique
K-means	Solution polyvalente, adaptée à diverses formes de données.
Mini-batch	Accélère le calcul, idéal pour de grandes quantités de données. Peut être moins précis de manière aléatoire.
Clustering agglomératif	Fournit une vue hiérarchique de la structure des clusters. Différents type de fusion possible afin de s'adapter au besoin.
DBSCAN	Identifie des clusters de forme arbitraire, sensible à la densité.
HDBSCAN	Identifie des clusters de manière robuste, adaptable aux densités variables.