

**NOM DU DOSSIER**

**Sous titre**

Date

Service

**RAPPORT DE TP**

**Oriane BERRY, Clémence LEMEILLEUR**

**Promo 56, Année 2022/2023 – 5SDBD-B1**

*« TP Apprentissage Non Supervisé »*

S1 2023

Encadrant : M.Siala

**RAPPORT DE TP**

Oriane Berry, Clémence LEMEILLEUR

Promo 56, Année 2022/2023 – 5SDBD-B1

*“TP Apprentissage Non Supervisé*”

S1 2023

Encadrant: M.SIALA

Table des matières

[Introduction 1](#_Toc116455969)

[I- Les points forts et faibles identifiés pour les différentes méthodes de Clustering 2](#_Toc116455970)

[1. Clustering k-Means et k- Medoids 2](#_Toc116455971)

[2. Clustering agglomératif 3](#_Toc116455972)

[3. Clustering DBSCAN et HDBSCAN 4](#_Toc116455973)

[4. Bilan 4](#_Toc116455974)

[II- Étude et comparaison de méthodes de clustering sur de nouvelles données fournies 4](#_Toc116455975)

[Conclusion 5](#_Toc116455976)

[Table des annexes 6](#_Toc116455977)

# *Introduction*

Dans ce TP nous avons comparé les différents algorithmes de clustering avec plusieurs méthodes qu’elles soient fournies par des outils ou externes. Nous utilisons pour cela des jeux de données en 2 dimensions afin d’avoir une meilleure visualisation de ces dernières.

En effet chaque méthode comporte des avantages, des inconvénients et fonctionne de manière plus ou moins efficace en fonction de l’aspect du jeu de données que nous voulons traiter.

Afin d’observer les effets de ces méthodes de clustering, nous utilisons des jeux de données en 2 dimensions afin d’avoir une meilleure visualisation de ces dernières. Nous finirons par les tester sur un dataset mystère afin de vérifier nos affirmations.

Nous avons travaillé sous Jupiter Notebook et héberger notre travail sur Github dont voici le lien : <https://github.com/Enario4/ApprentissageNonSupervise/> (Branche Main)

Tout d’abord nous allons brièvement rappeler :

Qu’est-ce que le clustering ?

Le clustering est une méthode d’apprentissage automatique qui à pour but de regrouper des points de données en fonction de leur similarité ou de leurs distances entre eux. Cette méthode d’apprentissage non supervisée est une technique populaire d’analyse statistique des données.

En prenant un ensemble donné de points, vous pouvez utiliser différents algorithmes de classification pour les regrouper dans des groupes spécifiques. Ils comporteront alors, dans un même groupe de points, des propriétés similaires tout en ayant des caractéristiques les différenciants de ceux des autres groupes.

Le but de la clusterisation est de donner un sens aux données, les faire parler afin de tirer un maximum d’informations qui, à la base, ne sont pas classées.

Cette méthode est utilisée avec des données de secteurs bien différents comme la santé, ou encore les études de profils.

Les manières de classer ses groupes varient en fonction de la forme des données. En effet, le les propriétés de chaque modèle vont influer sur le résultat. Parmi les principaux modèles on peut trouver ceux de groupe, centralisé, graphique, densité, distribué mais encore connectivité.

Nous allons maintenant étudier certaines méthodes afin de constater leurs avantages, inconvénients, et les comparer entre elles.

# Les points forts et faibles identifiés pour les différentes méthodes de Clustering

## Clustering k-Means et k- Medoids

Cette méthode est une des plus simple puisque le seul paramètre à choisir est k : le nombre de classes souhaité.

**Intérêts de la méthode k-Means**

***Les métriques d’évaluation recommandées :***

* **Coefficient de silhouette :** Différence entre la distance moyenne avec les points du même groupe que lui (cohésion) et la distance moyenne avec les points des autres groupes voisins (séparation). Le coefficient de silhouette varie entre -1 (pire classification) et 1 (meilleure classification).

**🡪** Est plus adapté aux données « regroupées » de manière identifiable.

* **Indice de Davies-Bouldin :** C'est la moyenne du rapport maximal entre la distance d'un point au centre de son groupe et la distance entre deux centres de groupes. Il varie entre 0 (meilleure classification) et +∞ (pire classification).

**🡪** Est plus adapté aux données reparties de manière homogène.

* **Indice de Calinski-Harabasz :** C'est le rapport entre la variance inter-groupes et la variance intra-groupe. Il varie entre 0 (pire classification) et +∞ (meilleure classification)
* **FasterPAM :** Permet d’évaluer les pertes que l’on a en coupant autour des médoïdes. Plus l’indice est petit mieux c’est.
* **Rand\_Score** : C’est une mesure de similarité entre deux partitions d'un ensemble. Sa valeur est comprise entre 0 et 1, 0 indiquant que les deux regroupements de données ne concordent sur aucune paire de points et 1 indiquant que les regroupements de données sont exactement les mêmes.
* **Mutual\_information**: Quantité qui mesure la dépendance entre 2 variables. Une information mutuelle élevée indique une forte dépendance ; et une information mutuelle nulle entre deux variables aléatoires signifie que les variables sont indépendantes.

Dans notre cas pour cette méthode, nous avons choisi la deuxième méthode. Après avoir remarqué que l’indice de DB n’est pas adapté à ce jeu de données. Nous avons testé sur l’indice de silhouette.

Nous avons ensuite choisi nos jeux de données avec ceux qui nous semblaient avoir des données bien séparées qui permette l’identification aisée des clusters par la méthode k-Means.

Lors de l’application itérative de la méthode des k-Means sur 2 jeux de données, voici les résultats que nous avons eu au niveau des métriques d’évaluation :

**JDD « aggregation.arff»**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 | k=8 | k=9 | k=10 |
| Indice de DB | **0,8964** | 0,6669 | 0,6116 | 0,7042 | 0,6400 | 0,7368 | 0,8270 | 0,7760 | 0,7252 |
| Indice silhouette | 0,4558 | 0,5234 | 0,5236 | 0,4987 | **0,6509** | 0,4804 | 0,4518 | 0,4621 | 0,4793 |
| Temps de calcul | 27,76 ms |  |  |  | 37,31 ms |  |  |  |  |
| Nb iter | 14 |  |  |  | 5 |  |  |  |  |

Nous allons faire de même avec un jeu de données du même aspect

**JDD « atom.arff»**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 | k=8 | k=9 | k=10 |
| Indice de DB | **1,0942** | 0,9038 | 0,7815 | 0,6566 | 0,6398 | 0,6553 | 0,6978 | 0,0722 | 0,7653 |
| Indice de silhouette | 0,4888 | 0,5358 | 0,5945 | 0,6352 | 0,6464 | **0,6499** | 0,6440 | 0,6372 | 0,5625 |
| Temps de calcul | 23,68ms |  |  |  |  | 39,46 ms |  |  |  |
| Nb iter | 4 |  |  |  |  | 6 |  |  |  |

**Limites de la méthode k-Means**

Pour mettre en avant les limites de cette méthode nous avons pris un autre jeu de données qui est complexe à clusteriser (forme de banane).

**JDD « banana.arff»**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 | k=8 | k=9 | k=10 |
| Indice de DB | 0,8930 | 0,8518 | 0,8083 | 0,6698 | 0,6567 | 0,6223 | 0,6141 | 0,5999 | 0,5931 |
| Indice de silhouette | 0,4645 | 0,4544 | 0,4623 | 0,4761 | 0,5054 | 0,5013 | 0,5218 | 0,5214 | 0,5201 |
| Temps de calcul | 34,33 | 49,57 | 48,83 | 58,49 | 52,26 | 60,69 | 88,74 | 72,34 | 71,49 |
| Nb iter | 7 | 5 | 17 | 14 | 12 | 8 | 15 | 11 | 12 |

**JDD «birch-rg1.arff»**

On remarque que l’exécution de ce code prend énormément de temps, et nous donne un résultat non satisfaisant. En effet il s’agit juste d’un amas de données coupées de manière équitables aléatoirement.

On remarque donc que la méthode k-Means comporte ses limites sur certaines formes de jeux de données, notamment celles dont les données sont très proches et il est difficiles de les séparer en plusieurs cluster.

**Méthode k-Medoids**

**JDD « aggregation.arff»**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 | k=8 | k=9 | k=10 |
| FasterPAM | 6672 | **4692** | 3827 | 3289 | 2929 | 2724 | 2556 | 2361 | 2233 |
| Indice de DB | 0,9208 | 0,7233 | 0,7429 | 0,8396 | 0,7446 | 0,7883 | 0,8570 | 0,8122 | 0,9022 |
| Indice de silhouette | 0,4546 | **0,5144** | 0,4930 | 0,4564 | 0,4828 | 0,4759 | 0,4260 | 0,4457 | 0,4028 |
| Rand\_score |  | **0,9832** |  |  |  |  |  |  |  |
| Mutual information |  | **1,0102** |  |  |  |  |  |  |  |
| Temps de calcul (ms) | 17,01 | 8,07 | 8,75 | 10,62 | 10,79 | 12,27 | 8,89 | 16,66 | 13,65 |
| Nb iter | 3 | 3 | 3 | 4 | 4 | 5 | 3 | 8 | 6 |

On remarque que pour k=3, on a un des pertes FasterPam pas trop élevées et un bon indice de silhouette. Le nombre de clusters le plus pertinent à choisir semble donc être 3.

De plus, on remarque qu’on obtient un rand score proche de 1 ce qui nous permet de déduire que les 2 méthodes, **k-Means et k-Medoïds**, clusterisent quasiment de la même façon ce jeu de données.

En revanche, l’indice d’information mutuelle est assez bas donc données sont très indépendantes et l’incertitude sur le résultat est d’autant plus grande.

Pour finir nous allons tester la méthode des k-medoids en utilisant la distance de Manhattan plutôt que la distance Euclidienne pour établir les clusters.

**JDD « aggregation.arff»**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 | k=8 | k=9 | k=10 |
| FasterPAM | 8305 | 5781 | 5138 | 4179 | 3817 | 3436 | 3183 | 3045 | 2770 |
| Indice de DB | 1,0240 | 0,7427 | 0,8739 | 0,7952 | 0,8630 | 0,7889 | 0,8457 | 0,8717 | 0,7663 |
| Indice de silhouette | 0,4120 | 0,4922 | 0,4116 | 0,4770 | 0,4488 | 0,4795 | 0,4415 | 0,4309 | 0,4681 |
| Temps de calcul (ms) | **5,38** | **5,26** | **5,64** | **6,46** | **5,31** | **7,52** | **9,28** | **5,18** | **6,08** |
| Nb iter | 3 | 3 | 4 | 4 | 5 | 4 | 5 | 4 | 5 |

On remarque plus de perte sur la distance de Manhattan, mais un meilleur indice de DB et surtout un temps de calcul bien plus avantageux.

## Clustering agglomératif

**Intérêts de la méthode**

Un des inconvénients des méthodes vues jusqu’ici est qu’il nous faut spécifier nous même le nombre de clusters souhaités : k.

Un des avantages de cette nouvelle méthode de clustering est que nous n’avons plus à spécifié ce paramètre.

Le clustering agglomératif va reconstruire les relations entre les clusters de manière arborescente. Une suite de méthodes fusionne au fur et à mesure les deux clusters les plus proches et ainsi de suite pour obtenir un dendrogramme, facile à observer.

Nous avons donc choisi 2 jeux de données de la sorte à ce qu’ils puissent identifier au mieux les clusters :

Nous pouvons faire varier plusieurs paramètres comme le seuil de distance, ou encore le nombre de cluster. Ici nous avons décidé de fixer le seuil de distance à None et faire varier le k.

**Paramètre : JDD « 2d-4c.arff » en single linkage**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K : | 2 | 3 | 4 | 5 | 6 | 7 |
| Nb de feuilles | 1261 | 1261 | 1261 | 1261 | 1261 | 1261 |
| Indice de DB | 0.4840 | 0.1877 | **0.1656** | 0.2085 | 0.2262 | 0.2650 |
| Indice Silhouette | 0.5601 | 0.8663 | 0.8670 | 0.7825 | 0.7335 | 0.6900 |
| Temps de calcul (ms) | 9.37 | 10.57 | 8.55 | 9.49 | 9.92 | 10.33 |

**Paramètre : JDD « hepta.arff» en single linkage**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K : | 2 | 3 | 4 | 5 | 6 | 7 |
| Nb de feuilles | 212 | 212 | 212 | 212 | 212 | 212 |
| Indice de DB | 0.6802 | 0.6867 | 0.5558 | **0.3321** | 0.4776 | 0.5181 |
| Indice Silhouette | 0.4044 | 0.4557 | 0.6070 | 0.7383 | 0.6893 | 0.6184 |
| Temps de calcul (ms) | 1.14 | 1.11 | 1.21 | 1.48 | 1.09 | 1.13 |

Nous avons ensuite voulu voir l’incidence des différentes manières de combiner les clusters (‘single’ comme précédemment, average, complete, ward linkage) et voici ce que nous avons pu observer :

**Paramètre : JDD « 2d-4c.arff », k=4**

|  |  |  |  |
| --- | --- | --- | --- |
| Linkage : | average | complete | ward |
| Nb de feuilles | 1261 | 1261 | 1261 |
| Indice de DB | 0.1656 | 0.1656 | 0.1656 |
| Indice Silhouette | 0.8670 | 0.8670 | 0.8670 |
| Temps de calcul (ms) | 24.82 | 25.27 | 31.26 |

**Paramètre : JDD « hepta.arff», k=5**

|  |  |  |  |
| --- | --- | --- | --- |
| Linkage : | average | complete | ward |
| Nb de feuilles | 1261 | 1261 | 1261 |
| Indice de DB | 0.3321 | 0.3321 | 0.3321 |
| Indice Silhouette | 0.7383 | 0.7383 | 0.7383 |
| Temps de calcul (ms) | 1.77 | 1.37 | 1.57 |

Nous pouvons donc en déduire que, le type de linkage à un impact sur le temps de calcul, notamment pour le jeu de données 1 où les temps sont multipliés par plus de 2.

**Limites de la méthode**

Maintenant que nous avons mieux compris comment fonctionnait cette méthode, nous l’avons appliqué à 2 jeux de données sur lesquels elle ne devrait pas fonctionner et voici ce que l’on a obtenu :

**Paramètre : JDD « dpb.arff » en single linkage**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| K : | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Nb de feuilles | 4000 | 4000 | 4000 | 4000 | 4000 | 4000 | 4000 |
| Indice de DB | 0.4585 | 0.4396 | 0.4797 | 0.4702 | 0.4628 | **0.4527** | 0.4562 |
| Indice Silhouette | 0.3290 | 0.2551 | 0.1863 | 0.1857 | 0.1497 | 0.1460 | 0.1447 |
| Temps de calcul (ms) | 68.63 | 73.6 | 66.45 | 72.0 | 64.79 | 66.61 | 63.9 |

**Paramètre : JDD « cluto-t7-10k.arff » en single linkage**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| K : | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Nb de feuilles | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 |
| Indice de DB | **0.6289** | 0.8762 | 0.8054 | 0.8335 | 0.7968 | 2.4551 | 2.2946 |
| Indice Silhouette | 0.1671 | -0.064 | -0.2429 | -0,2474 | -0.3243 | -0.4782 | **-0.5335** |
| Temps de calcul (ms) | 564.69 | 551.68 | 553.92 | 557.08 | 571.77 | 562.78 | 543.21 |

Nous avons fait le choix de le faire uniquement en single linkage pour faciliter les comparaisons avec les résultats au-dessus.

On remarque que les résultats sont bien moins satisfaisants que pour les jeux de données précédent et que les temps d’exécution sont énormément plus grands. Les différentes données étant trop rapprochées, l’algorithme a du mal à les regrouper en établissant des limites.

**Comparaison de méthodes de clustering**

Nous allons étudier cette comparaison en détail dans la partie 2 de ce rapport.

## Clustering DBSCAN et HDBSCAN

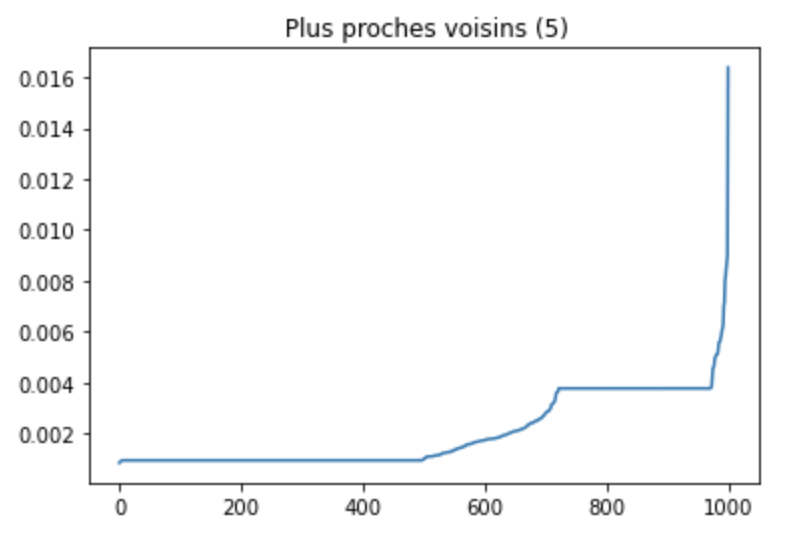
**Intérêts de la méthode DBSCAN**

Les méthodes DBSCAN et HDBSCAN sont des méthodes de densité. Les classes représentent donc des zones de fortes densités de points en comparaison des autres zones.

Un des avantages de la méthode DBSCAN est qu’elle est capable de détecter ce qu’on pourrait qualifier de « bruit », les points trop éloignés et faussant le reste des données.

Nous avons donc choisi 2 jeux de données de la sorte à ce qu’ils puissent identifier au mieux les clusters. Nous avons laissé la distance en valeur par défaut, fixé min-sample à 10 et fait varier le paramètre eps.

Pour choisir l’échelle de variation de eps nous nous basons sur le nombre de plus proche voisin et nous prénoms entre le moment ou la courbe est assez stable jusqu’au moment où le pic est bien avancé.



Par exemple ici nous prendrons entre 0.002 et 0.008

**Paramètre : JDD « donut3.arff » min\_samples=10**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| eps | 0.002 | 0.003 | 0.004 | 0.005 | 0.006 | 0.007 | 0.008 |
| Indice de DB | 1.7304 | 2.1348 | 2.3620 | 2.5070 | 2.5921 | **0.6621** | 2.7050 |
| Indice de silhouette | -0.5873 | -0,3048 | 0.2184 | 0.3038 | 0.3435 | 0.3651 | 0.3776 |
| Temps de calcul (ms) | 3.98 | 6.3 | 4.5 | 6.2 | 6.47 | 6.5 | 6.82 |
| k | 2 | 11 | 2 | 2 | 2 | 2 | 2 |
| Nb bruit | 977 | 596 | 470 | 406 | 375 | 357 | 346 |

**Paramètre : JDD « donutcurves.arff » min\_samples=10**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| eps | 0.002 | 0.003 | 0.004 | 0.005 | 0.006 | 0.007 | 0.008 |
| Indice de DB | / | **1.0245** | 14.9767 | 26.7422 | 21/6669 | 24.3115 | 26.5387 |
| Indice de silhouette | / | **0.0175** | 0.1109 | 0.2826 | 0.2990 | 0.3112 | 0.3125 |
| Temps de calcul (ms) | / | 4.51 | 4.07 | 4.63 | 4.95 | 4.83 | 5.3 |
| k | / | 1 | 4 | 3 | 3 | 3 | 3 |
| Nb bruit | / | 910 | 307 | 290 | 270 | 261 | 260 |

Pour le paramètre de min\_samples= 10, on peut donc dire que pour le premier jeu de données le paramètres eps optimal est de 0.007 et de 0.003 pour le deuxième.

Il est possible de faire la même démarche en fixant eps et en faisant varier min\_samples pour trouver sa valeur optimale.

**Limites de la méthode DBSCAN**

Maintenant que nous avons mieux compris comment fonctionnait cette méthode, nous l’avons appliqué à 1 jeu de données sur lesquels elle ne devrait pas fonctionner et voici ce que l’on a obtenu :

**Paramètre : JDD « impossible.arff »**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| eps | 0.15 | 0.25 | 0.5 | 1 |
| Indice de DB | 1.1670 | 1.0958 | 17.8940 | 1.7189 |
| Indice de silhouette | **-0.4355** | -0.0187 | 0.0665 | 0.5900 |
| Temps de calcul (ms) | 13.89 | 15.42 | 23.92 | 32.64 |
| k | 31 | 28 | 12 | 4 |
| Nb bruit | 2694 | 1451 | 438 | 57 |

Nous pouvons donc voir que la méthode DBSCAN n’est pas adapté à ce genre de jeu de données car cette méthode identifie des clusters avec une forme quelconques, donc si la forme en diffère-t-elle ne sera pas reconnue. Le temps d’exécution est d’ailleurs bien plus élévé.

**Comparaison avec la méthode HDBSCAN**

Nous allons rapidement faire un petit point sur la comparaison entre ces 2 méthodes avant d’élargir notre vision en comparant, de manière globale, toutes les méthodes de clustering entre elles.

La méthode HDBSCAN est connue pour être insensible à la variabilité de densité dans les données.

Nous avons repris les JDD précédent en les traitants avec HDBSCAN.

**Paramètre : JDD « donut3.arff » min\_samples=10**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Min samples | 10 | 20 | 30 | 40 |
| Indice de DB | 2.7973 | 2.7973 | 2.7973 | 2.78.07 |
| Indice de silhouette | 0.3899 | 0.3899 | 0.3899 | 0.3886 |
| Temps de calcul (ms) | **15.05** | 22.48 | 16.42 | 17.35 |
| k | 3 | 3 | 3 | 2 |
| Nb bruit | 0 | 0 | 0 | 335 |

**Paramètre : JDD « donutcurves.arff » min\_samples=10**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Min samples | 10 | 20 | 30 | 40 |
| Indice de DB | 33.3075 | 33.3075 | 43.6224 | 43.6224 |
| Indice de silhouette | 0.3249 | 0.3249 | 0.5243 | 0.5243 |
| Temps de calcul (ms) | **14.66** | 15.1 | 14.88 | 15.4 |
| k | 4 | 4 | 3 | 3 |
| Nb bruit | 0 | 0 | 0 | 0 |

Les valeurs des paramètres qui ressortent ici sont Min samples = 10 pour le premier JDD et Min samples = 10 pour le deuxième jeu de donnée.

Nous remarquons d’ailleurs que la variation de Min samples ne change pas énormément les valeurs des indices.

La méthode HDBSCAN ne comporte plus qu’un paramètre à faire varier donc plus optimale à gérer. De plus le bruit est énormément réduit par rapport à HDBSCAN en revanche les performances de temps de calcul sont moins bonnes…

# Étude et comparaison de méthodes de clustering sur de nouvelles données fournies

Après avoir étudier les différentes méthodes, leurs points forts et points faibles respectifs, nous allons les tester sur un nouveau dataset : le dataset mystère et observer leur comportement.

Pour cela nous allons les tester sur un nouveau jeu de données fournies, s’apparentant à un cas que nous pourrions retrouver dans le cadre d’une étude réelle. Nous avons utilisé pour cela un serveur de calcul.

Cela va nous permettre de voir si les caractéristiques mises en avant précédemment sont vérifiées ou démenties.

Nous illustrerons toutes nos observations à l’aide de visualisation pour rendre cette comparaison plus parlante.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **k-Means** | **k-medoids** | **Clustering agglomératif** | **DBSCAN** | **HDBSCAN** |
| **Qualité des solutions obtenues** |  |  |  |  |  |
| **Performances des méthodes** |  |  |  |  |  |
| **Type de Dataset adapté** |  |  |  |  |  |

# *Conclusion*

Pour conclure nous pouvons dire que ce TP nous a permis de prendre connaissance et de nous familiariser avec différentes méthodes de clustering et de visualiser leurs effets sur des jeux de données en 2 dimensions.

Nous avons pu expérimenter et mieux comprendre les principes de base du clustering de manière globale dans un premier temps, puis plus approfondis via différentes méthodes en comparant leurs avantages et inconvénients.

Cela nous permet donc de nous mettre une fois de plus dans le rôle de l’ingénieur qui est d’utiliser ses connaissances et de les appliquer à des cas réels. D’autant plus que le clustering peut être appliqué à énormément de cas d’usages dans de nombreux domaines et, selon nous, est chaque jour un peu plus dans l’actualité.

# Table des annexes

1. Annexe 1 : Lien GitHubA

**Annexe 1 :** Lien GitHub : <https://github.com/Enario4/ApprentissageNonSupervise> (Branche Main)

