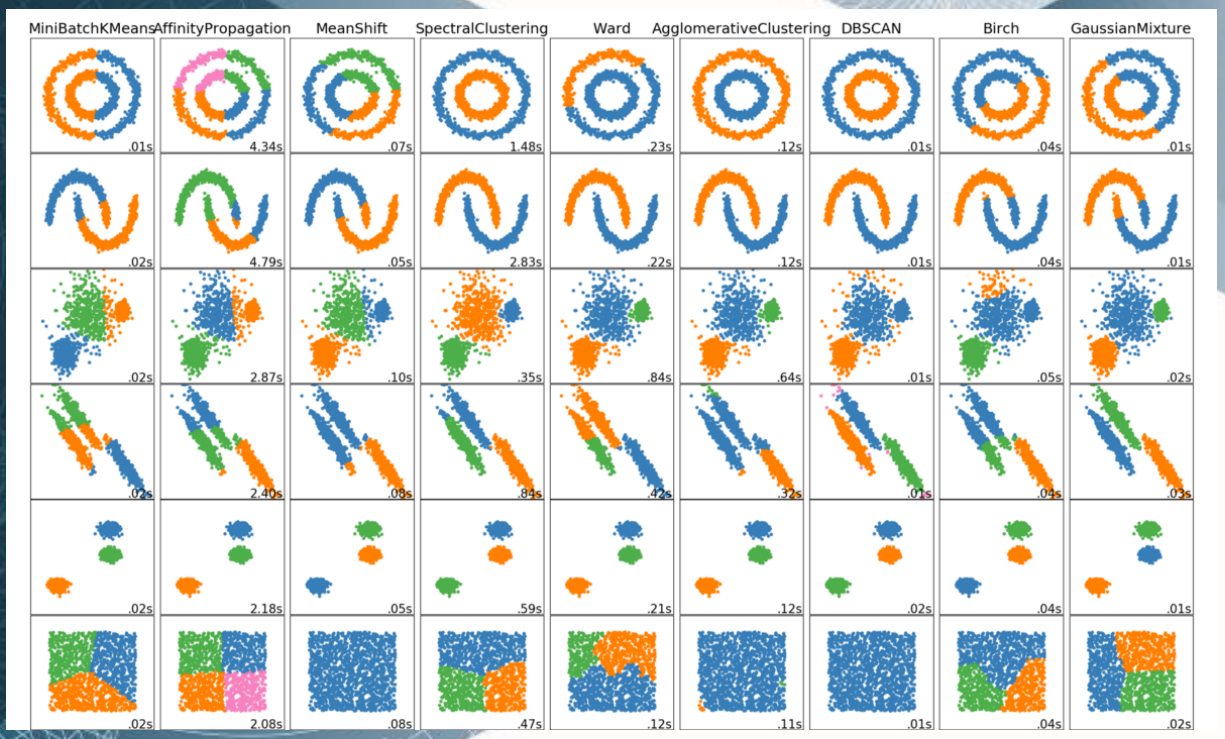
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE



Apprentissage Non Supervisé

Rapport TP - Clustering

5-SDBD

Semestre 1 année 2022/2023

Rouby Nathan Soumaré Coumba – Groupe 5SDBD-B2

Apprentissage Non Supervisé

Rapport TP - Clustering

5-SDBD

Semestre 1 année 2022/2023

Rouby Nathan Soumaré Coumba – Groupe 5SDBD-B2

TABLE DES MATIERES

----------INTRODUCTION ----------

Dans le cadre du cours d'Analyse descriptive et prédictive de ce premier semestre de cinquième année Systèmes Distribués et Big Data, nous nous sommes penchés sur l'étude de plusieurs jeux de données afin de tester et comparer différents algorithmes de clustering.

L'idée était d'appréhender les différents algorithmes sur des jeux de données tests connus afin d'ensuite appliquer les mêmes algorithmes sur des jeux de données à analyser.

Notre projet peut se retrouver sur un dépôt git au lien suivant : https://github.com/cosoumare/Clustering

1 METHODES DE CLUSTERING

1.1 CLUSTERING K-MEANS ET K-MEDOIDS

Dans cette première partie du TP, nous nous basons sur des jeux de données fournies à l'adresse suivantes https://github.com/deric/clustering-benchmark afin d'analyser les méthodes.

1.1.1 Méthode k-Means

La méthode k-Means correspond à un algorithme de clustering non hiérarchique, c'est-à-dire que la création de clusters ne se fait pas dans un ordre défini mais par séparation ou fusion de clusters succinctes en minimisant ou maximisant un ou des critères spécifiques. Dans le cas de cette méthode, l'objectif est de minimiser la distance intra-cluster en récupérant en entrée le nombre de clusters à obtenir.

Cet algorithme commence par la mise en place d'autant de centres de gravité que de nombre de clusters que l'on souhaite obtenir. Chaque centre de gravité (pas forcément un élément du jeu de données) est affecté à un cluster.

A chaque boucle de l'algorithme on commencera par affecter chaque élément du jeu de données au centre de gravité le plus proche avant de réévaluer les centres de gravité de chacun des clusters ainsi créés, et ceci jusqu'à ce que les conditions d'arrêt soient valides (nombres d'itérations max, centres de gravités stables).

Pour affecter chaque élément à un cluster le plus proche on choisit une mesure de distance euclidienne.

1.1.2 Intérêts de la méthode k-Means

On choisit ici des jeux de données afin d'évaluer notre méthode k-Means appliquée via la méthode KMeans du package sklearn.

A première vue on choisit des jeux de données dans lesquels on fixe le nombre de clusters idéals que l'on souhaite obtenir.

On obtient par exemple le résultat ci-contre en appliquant la méthode au jeu de données xclara.

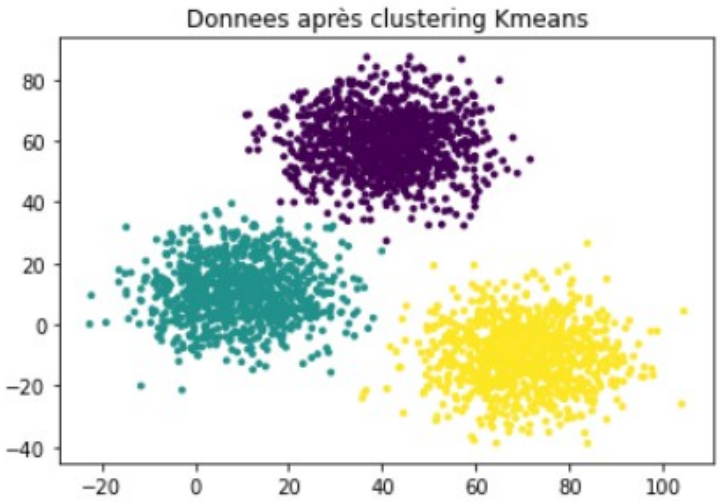


Figure 1: Résultats méthode kMeans sur jeu de données xclara (Tps de calcul : 14,6ms)

On observe ici les avantages de la méthode kMeans sur ce jeu de données. En effet, celui-ci présente des éléments facilement identifiables en termes de distances dans chacun des clusters. L'algorithme permet une **facilité d'interprétation** couplée à sa **simplicité d'implémentation** et son **efficacité**. Il est particulièrement fonctionnel sur des datasets contenant un grand nombre de données et pour les datasets contenant des clusters sphériques.

Il possède également un **faible coût de calcul**. En effet, l'algorithme est de complexité NP-difficile : O(n*d*k) avec n le nombre d'itérations, d la distance de référence et k le nombre de clusters.

En appliquant les métriques d'évaluation sur le même jeu de données, on obtient les résultats suivants :

- *Silhouette coeff : 0.69*

Compris entre [-1,1], ce coefficient combine 2 mesures : la distance moyenne entre chaque élément et le reste des éléments de son cluster ainsi que la distance moyenne minimale de l'élément aux éléments des autres clusters.

Ici, on reste loin de 1 qui correspond au score pour des clusters très denses et bien séparés.

- *Davies Bouldin : 0.42*

Compris entre [0,1], ce coefficient s'appuie sur la moyenne des distances entre chaque élément et le centre de gravité de son cluster ainsi que sur la distance entre les centres des clusters.

On souhaite ici minimiser ce score afin d'avoir des clusters homogènes et séparés.

- *Calinski Harabasz : 10827*

Compris entre [0, +inf], ce coefficient correspond au rapport entre la somme de la dispersion entre éléments du même cluster et la somme de la dispersion des éléments de clusters différents. Ce coefficient est particulièrement sensible à la taille du jeu de données.

On souhaite ici maximiser au maximum ce coefficient.

En utilisant ces métriques, on souhaite automatiser notre initialisation du nombre de clusters en début d'algorithme.

En utilisant la métrique silhouette, on laisse tourner notre algorithme en augmentant progressivement le nombre de cluster en stoppant les itérations lorsque la métrique d'évaluation perd en performance. Tant que la métrique donne un résultat de performance de plus en plus positif on continue à augmenter le nombre de cluster.

1.1.3 Limites de la méthode k-Means

- L'initialisation a un impact sur le développement de l'algorithme. Les centres des clusters choisis avant le lancement de l'algorithme auront un impact sur les clusters définis par la suite. De plus, l'initialisation requiert la définition du nombre de clusters a recherché, sans effectuer l'amélioration ajoutée précédemment, on remarquait l'évolution des métriques en fonction du nombre de clusters demandé à l'initialisation.
- L'algorithme ne fonctionne pas avec des clusters de formes non sphériques. On teste par exemple l'algorithme avec des clusters différents :

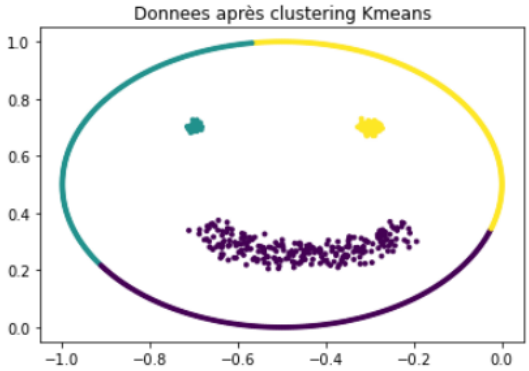


Figure 2: Résultat k-Means sur le jeu de données Smile3

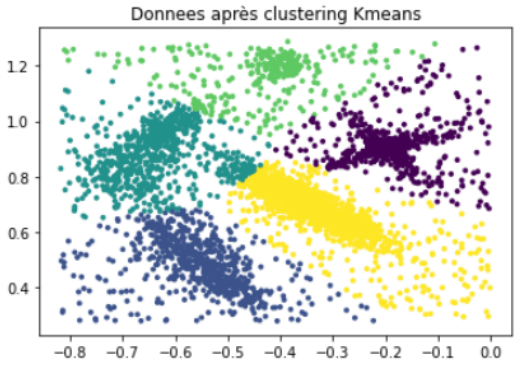


Figure 3: Résultat k-Means sur le jeu de données dpb

- Ici, on voit également que la métrique est plus performante pour un nombre de clusters à 3. L'identification du nombre de clusters n'est également pas toujours précise pour la méthode k-Means avec la métrique utilisée.

- Enfin la distance utilisée pour la formation des clusters peut également entraîner des mauvais découpages dans les clusters finaux.

1.1.4 Méthode k-Medoids

Dans le cas de la méthode k-Medoids, le point central de chaque cluster correspond à un élément du jeu de donné. Celui-ci est le point dont la dissimilarité avec tous les autres points est la minimale.

Dans cette méthode on détermine alors dès l'initialisation les représentants initiaux afin d'affecter par la suite chaque élément du dataset à son représentant le plus proche.

En testant notre méthode avec le dataset xclara testé avec la méthode k-Means, on obtient les résultats suivants.

- Temps de calcul en ms : xclara : 136ms. Le temps de calcul est largement plus important que pour la méthode k-Means, il est d'autant plus important que le jeu de données est grand.
- On compare nos méthodes via rand_score. Cette méthode fournit un indice de similarité entre les deux algorithmes en utilisant le nombre d'accords et de désaccords selon des pairs d'éléments (pairs dans le même cluster dans K-Means & K-Medoids / pairs dans des clusters différents dans K-Means & K-Medoids)

  Comparaison des méthodes kmeans/kmedoids via rand_score :
  - xclara : 1 (similarité optimale)
  - hepta : 1
  - square2 : 0.994
  - smile3 : 0.978

  Cependant, en changeant la distance euclidienne par la distance de manhattan, on remarque une diminution du rand_score en comparaison avec la méthode kmeans (Sensibilité des méthodes à méthode de distance utilisée).

1.2 CLUSTERING AGGLOMERATIF

1.2.1 Méthode et principe

1.2.2 Intérêts et limites de la méthode

La méthode du clustering agglomératif nous donne une réelle flexibilité, car les résultats sont drastiquement différents selon le linkage choisi. Par exemple, en limitant le nombre de clusters, 'average' et 'ward' nous donnent des résultats similaires aux méthodes précédentes (même si les proportions peuvent changer). En revanche, 'single' linkage change totalement les résultats, permettant par exemple de séparer correctement les clusters de smile3, mais se retrouve en échec sur des clusters jusqu'ici bien traités, comme square2.
Ce 'single linkage' est très intéressant dans les datasets où les clusters sont très denses, même s'ils prennent beaucoup d'espace avec des formes sphériques ou convexes. Cependant, il est inefficace dans les datasets où la densité d'un même cluster est moins élevée, ou dans ceux dont les clusters sont trop proches les uns des autres.

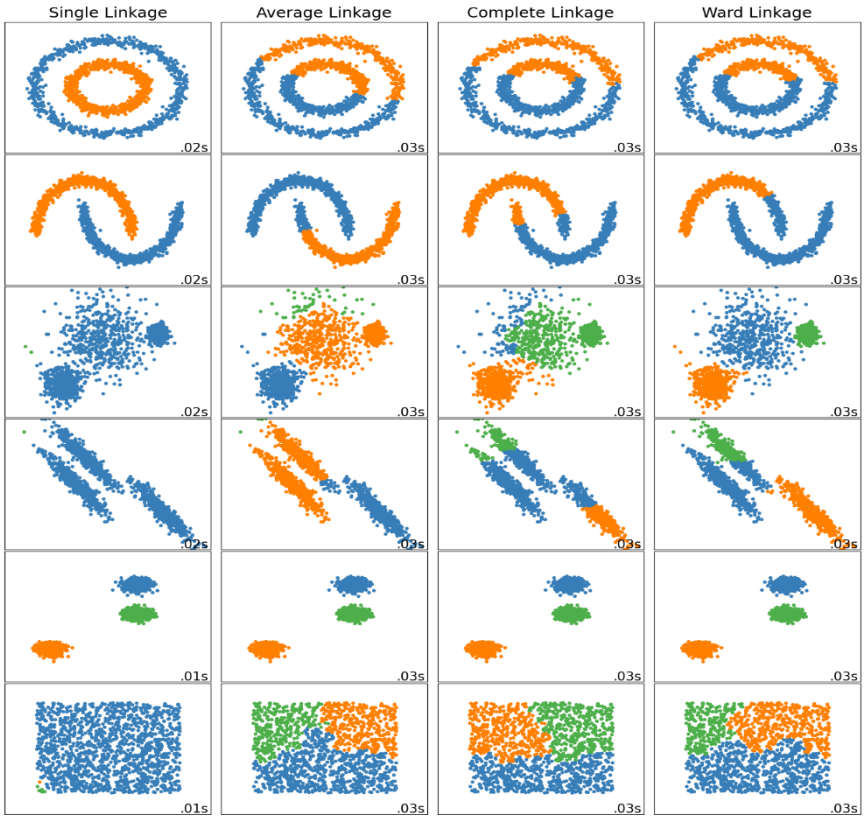
Les temps de calcul dépendent de la limite imposée, mais restent raisonnables si celle-ci est choisie convenablement.

1.2.3 Comparaison des méthodes k-Means, k-Medoids et clustering agglomératif

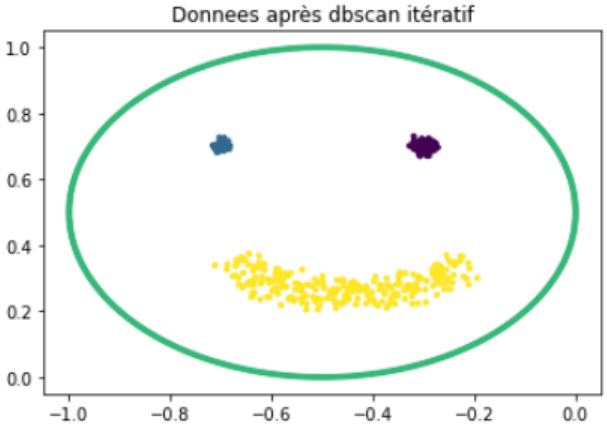
La méthode clustering agglomératif avec une limite en nombre de clusters est la première du TP à pouvoir bien fonctionner sur smile3 (avec 4 clusters), tout en maintenant la possibilité de traiter les autres, comme xclara ou 2d-4c, en choisissant les variables adéquates.

Nous remarquons que le silhouette_score fonctionne très mal sur des datasets comme smile3, et ce quelle que soit la méthode choisie.

	xclara	square2	hepta	smile3	s-set3	dpb
kmeans	14 ms	103	88	94	991	46
kmedoids	136	30	1	22	290	250
agg (nbr)	119 (complete)	10 (complete)	1 (complete)	11 (single)	360 (complete)	192 (complete)



1.3 METHODES DBSCAN ET HDBSCAN :



DBSCAN est très efficace pour séparer les datasets dont les clusters sont denses, y compris s'ils sont convexes. A ce titre, les datasets comme smile3 sont idéaux, mais même ceux comme xclara fournissent de bons résultats une fois la bonne valeur d'epsilon choisie.
Cependant, lorsque les clusters sont proches les uns des autres, il semble inaltérable que les points les plus éloignés de leur cluster respectif soient isolés par DBSCAN, car si on augmente trop la valeur d'epsilon, tous les points se retrouvent dans le même cluster. Au même titre, si les clusters ont des densités différentes, DBSCAN ne peut les gérer correctement.
Les temps de calculs sont très faibles, mais peuvent augmenter drastiquement avec la taille du dataset.

HDBSCAN est designé pour régler ces problèmes. Comme on pouvait s'y attendre, cette méthode offre d'excellents résultats sur smile3 et xclara, comme DBSCAN. Cependant, bien que les performances sur square2 soient infiniment meilleures que celles de DBSCAN, elles restent décevantes comparé à d'autres méthodes testées précédemment. De même, mais pour différentes raisons, ses résultats sur s-set3 sont peu encourageants. Cette méthode offre cependant des résultats uniques sur les datasets comme dpb, où les multiples points isolés sont rassemblés dans un grand cluster.
Le grand avantage de cette solution, outre le temps de calcul tout à fait acceptable, est de ne pas avoir à la guider avec des variables : en effet, la seule nécessité est d'imposer le nombre de points minimal d'un cluster, ce qui est souvent bien plus aisé que déterminer le nombre de clusters ou la distance minimale entre eux.

2 ETUDE ET ANALYSE COMPARATIVE DE METHODES DE CLUSTERING

2.1 ANALYSE INDIVIDUELLE JEU DE DONNEES

2.1.1 x1.txt

En appliquant la méthode kMeans au premier dataset, on obtient le résultat suivant :

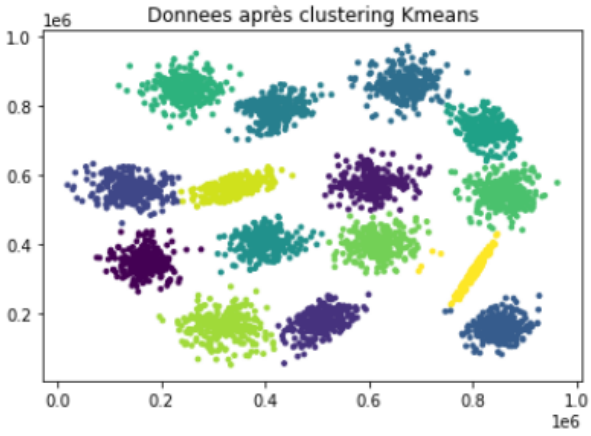


Figure 4 : Résultat kMeans pour le dataset x1.txt

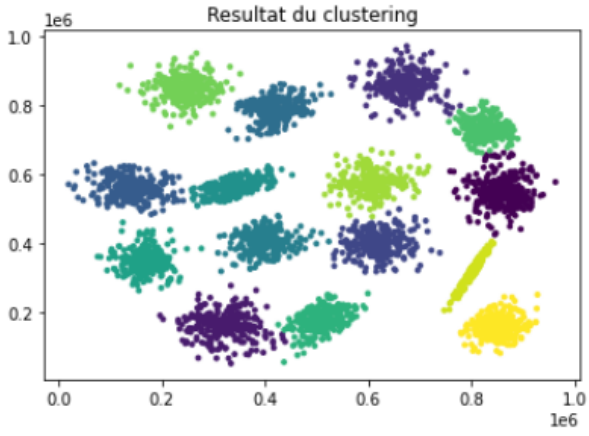


Figure 3: Résultat clustering agglomératif "ward" dataset x1.txt

On a ici un dataset composé de 15 clusters sphériques, la méthode **kMeans** permet donc une séparation des clusters plutôt performante, bien que plus lente que les méthodes de clustering agglomeratif, dbscan et hdbscan. Même résultat, mais encore plus lent, nous avons la méthode **kmedoids**.

En ce qui concerne le **clustering agglomératif** :
x1 est parfaitement séparé avec les linkage 'average' et 'ward' (limite : 15 clusters) : résultats identiques et "visuellement" parfaits.
En termes d'études par les coefficients silhouette, David-Bouldin et Calinski-Harabasz, les résultats en utilisant les différents linkage semblent très proches, quasiment similaires (hormis single, cf partie 1).

La méthode **dbscan** donne un résultat bien moins précis au niveau des irrégularités contrairement au clustering agglomératif ou même à la méthode k-Means. Cependant, au niveau du temps d'exécution, la méthode dbscan est de loin la plus efficace. Les éléments étant extrêmement rapprochés au sein des clusters, on doit appliquer la méthode dbscan avec un epsilon très faible (ici résultat avec un epsilon égal à $0.15.10^{e}7$ pour min_samples=5).

La méthode **hdbscan**, enfin, semble être un entre-deux entre la rapidité de dbscan et la performance des autres méthodes.

Tableau comparatif des résultats de chaque méthode pour le dataset x1 :

Algorithme	Temps d'exécution (en ms)	Coefficient *Silhouette*	Coefficient *Davies-Bouldin*	Coefficient *Calinski Harabasz*
k-means	183.98	0.711	0.37	22680
k-medoids	722.04	0.711	0.37	22679
agglomératif (single)	294.12	-0.049	0.93	1147
aggl. (average)	1295.38	0.708	0.37	22298
aggl.(complete)	964.26	0.701	0.37	21330
aggl (ward)	1459.14	0.709	0.36	22330
dbscan	55.97	0.505	1.37	2431
hdbscan	81.74	0.607	1.50	3300

2.1.2 x2.txt

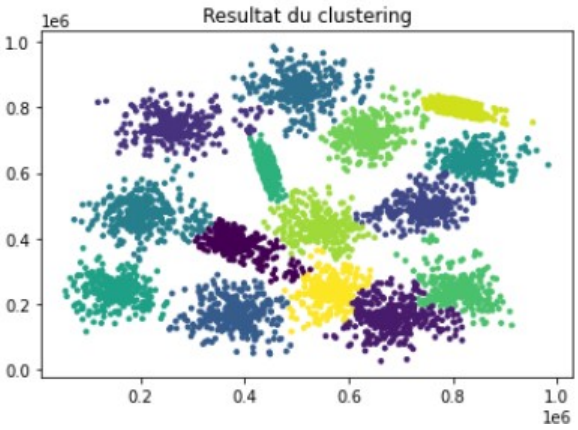
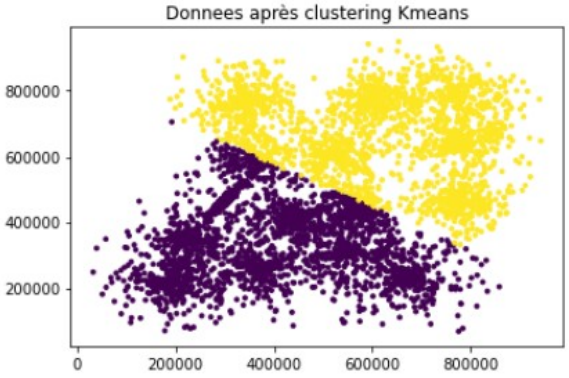
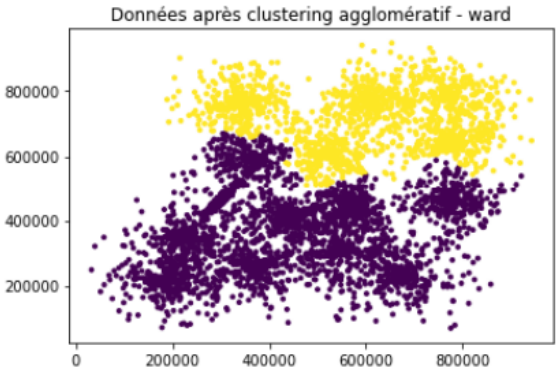


Figure 5 : Résultat clustering agglomératif "ward" dataset x2

Comme pour le dataset x1, le dataset x2 contient des éléments très proches et nombreux, ce qui explique la difficulté de séparation en cluster pour l'ensemble des méthodes.
x2 est convenablement séparé avec kmeans, kmedoids, et le clustering agglomératif avec les linkages 'average' et 'ward' (limite : 15 clusters). De légères différences subsistent, concernant quelques points dont l'attribution "visuelle" génère une hésitation même chez l'humain.
Les autres méthodes ont beaucoup de mal à séparer les clusters, car ils sont très proches les uns des autres.

2.1.3 x3.txt





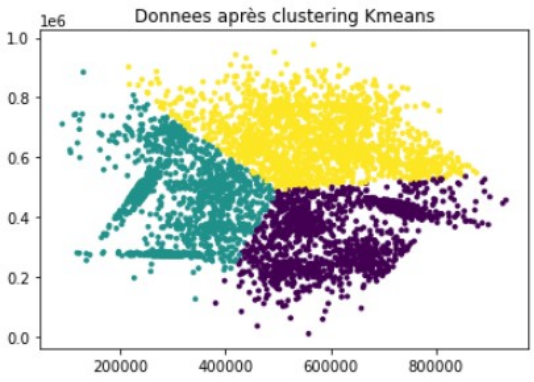
Ici, x3 possède une forte densité d'éléments peu séparés. Dès l'application de la méthode kMeans, on remarque qu'aucun découpage visuel apparaît. Le découpage en 2 clusters est celui donnant de meilleures performances selon les différents indices testé (silhouette, David-Bouldin, Calinski-Harabasz), cependant visuellement les résultats ne sont pas satisfaisants.

En appliquant le clustering agglomératif, x3 ne présente aucun découpage convenable, on a différentes erreurs pour tous les linkages.

Tableau comparatif des résultats de chaque méthode pour le dataset x3 :

Algorithme	Temps d'exécution (en ms)	Coefficient *Silhouette*	Coefficient *Davies-Bouldin*	Coefficient *Calinski Harabasz*
k-means	103.99	0.407	0.99	4292
k-medoids	492.68	0.409	0.98	4259
agglomératif (single)	492.55	0.306	0.51	3
aggl. (average)	959.93	0.338	1.16	2852
aggl.(complete)	951.92	0.396	1.01	4124
aggl (ward)	1503.58	0.383	0.996	3503
Dbscan (esp=0.0163^{e}6 / min_samples = 8)	80.00	-0.412	2.13	218
hdbscan	157.58	-0.053	1.51	186

2.1.4 x4.txt



Dans le cas du dataset x4 on a un set extrêmement dense, on ne peut même pas déterminer les différents clusters visuellement. Il nous semble encore une fois qu'aucune méthode ne parvienne à découper ce dataset en clusters qui fassent sens.
En effet, dans ce cas-ci les éléments sont tous bien trop proches mis-à-part quelques singularités.

2.1.5 y1.txt

Le dataset est trop lourd pour que nous puissions l'analyser, nous obtenons malheureusement une Memory Error.

2.1.6 zz1.txt

Dans le cas du dataset zz1.txt, on remarque deux différents types de clusters. Des clusters dont les éléments sont visuellement très proches et d'autres dont les éléments sont sensiblement plus dispersés.

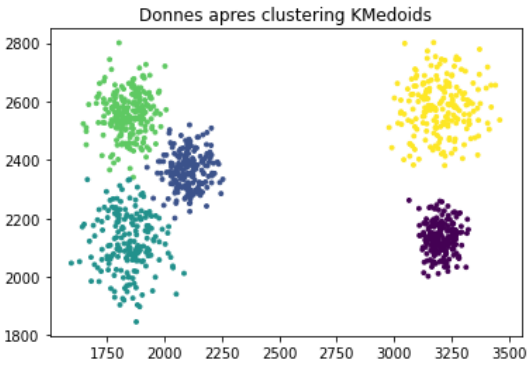
En ce qui concerne le clustering agglomératif, seuls les linkage average et ward semblent visuellement parfaits (limite : 8 clusters). K-means, k-medoids et dbscan ont eu du mal à séparer les clusters du milieu, proches entre eux mais dont la densité variait.

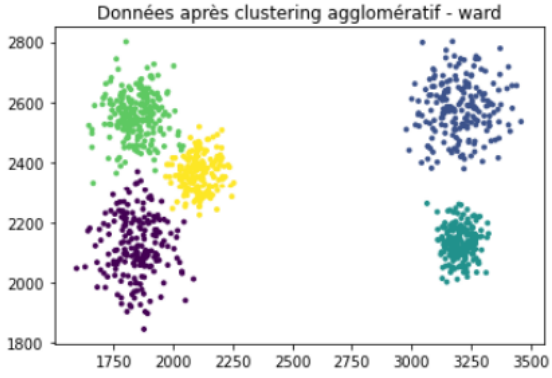
Etrangement, nous remarquons une augmentation massive du temps d'exécution avec DBSCAN sur ce dataset uniquement et ce quelles que soient les valeurs utilisées pour les paramètres, mais nous ne parvenons pas à l'expliquer.

Tableau comparatif des résultats de chaque méthode pour le dataset zz1 :

Algorithme	Temps d'exécution (en ms)	Coefficient *Silhouette*	Coefficient *Davies-Bouldin*	Coefficient *Calinski Harabasz*
k-means	140.92	0.846	0.39	41896
k-medoids	861.7	0.835	0.41	30413
agglomératif (single)	16.99	0.110	0.58	1442
aggl. (average)	32.98	0.504	0.54	5442
aggl.(complete)	35.98	0.426	1.07	6131
aggl (ward)	36.97	0.518	0.84	7023
Dbscan (esp= 0.01661e6 / min_samples = 5)	350.80	0.418	0.45	0.45
hdbscan	212.39	0.840	2.27	22077

2.1.7 zz2.txt





La plupart des méthodes ont bien découpé le dataset, d'où les résultats identiques en ce qui concerne les métriques d'évaluation. En ce qui concerne le temps d'exécution, c'est

étrangement k-medoids qui a le meilleur temps parmi les séparations parfaites. Dbscan n'a pas su séparer les clusters de gauche sur l'image, trop proches entre eux, tandis qu'hdbscan offre une nouvelle fois un entre-deux entre la performance et la rapidité.

Tableau comparatif des résultats de chaque méthode pour le dataset zz2 :

Algorithme	Temps d'exécution (en ms)	Coefficient *Silhouette*	Coefficient *Davies-Bouldin*	Coefficient *Calinski Harabasz*
k-means	56.00	0.769	0.36	6433
k-medoids	20.94	0.769	0.36	6433
agglomératif (single)	32.00	0.769	0.36	6433
aggl. (average)	55.97	0.769	0.36	6433
aggl.(complete)	34.98	0.769	0.36	6433
aggl (ward)	36.98	0.769	0.36	6433
Dbscan (esp=15 / min_samples = 5)	7.99	-0.443	1.31	21
hdbscan	17.95	0.586	1.33	5529

2.2 BILAN COMPARATIF DES METHODES

Pour la méthode **k-Means**, on a pu noter des avantages certains dans le cas des datasets x1 et zz1 pour des clusters majoritairement bien délimités. Mais globalement c'est l'une des méthodes qui nous donnent les résultats les plus décevants lorsque l'on se place avec des sets dont les clusters sont trop proches les uns des autres ou avec des formes particulières.

La méthode **k-Medoids** est plus robuste que k-Means, cependant elle est souvent plus lente et n'offre de solution qu'à une seule de limites de k-Means (la densité des clusters) alors que ces méthodes ont des faiblesses sur

Pour le **clustering agglomératif**, le linkage 'single' n'était jamais pertinent pour les datasets étudiés (cf Partie 1), tandis que 'complete' donnait la plupart du temps des résultats comparables à 'average' et à 'ward', mais systématiquement moins précis. Ces deux derniers proposaient souvent des découpages du dataset utilisables, avec cependant parfois des temps d'exécutions assez longs.

En ce qui concerne **DBSCAN**, nous n'avons pas réussi à trouver des valeurs de variables qui séparaient les clusters avec assez de précision. C'est la méthode avec le meilleur temps d'exécution et de loin, mais la précision avec laquelle on doit guider ses paramètres pour obtenir de bons résultats nous a empêché d'utiliser son plein potentiel.

HDBSCAN nous a donné des résultats "convenables" assez rapidement sur les datasets zz1 et 2, même s'ils étaient finalement moins bons que ceux obtenus via d'autres méthodes plus lentes. Son gros avantage reste tout de même que ces résultats sont obtenus quasiment sans aucun guide humain dans les paramètres, contrairement aux autres méthodes.

----------CONCLUSION----------

Dans ce TP nous avons ainsi pu découvrir et comparer différentes méthodes de clustering notamment des méthodes furnies par scikitlearn. L'étude des méthodes k-means, k-medoids, clustering agglomératif, dbscan et hdbscan à travers leurs applications sur multiples datasets nous a permis de nous familiariser avec un concept du machine learning, l'apprentissage non supervisé.

Cette étude nous a notamment montré que les performances de chacune des méthodes de partitionnement sont entièrement dépendantes du dataset étudié et que le choix de la méthode doit être effectué en fonction de l'utilisation voulue des données.

INSA Toulouse
135, avenue de Rangueil
31077 Toulouse Cedex 4 - France
www.insa-toulouse.fr

Université Fédérale Toulouse Midi-Pyrénées
Liberté • Égalité • Fraternité RÉPUBLIQUE FRANÇAISE
MINISTÈRE DE L'ÉDUCATION NATIONALE, DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE