Yosra ZEYRI NEMRI – MS IoT
Lien GitHub : https://github.com/yosraZEYRINEMRI/MSIoT-Big-Data

TP Clustering

Objectif

L'objectif de ces TP est de comparer différentes techniques de clustering faisant partie de la librairie scikit-learn.

- K-means
- Clustering Hiérarchique (agglomératif)
- DB-SCAN

Jeux de donnée

Nous allons utiliser des jeux de données open-source largement utilisées dans le benchmarking de techniques de clustering. Ces derniers figurent dans le repo GitHub suivant :

https://github.com/deric/clustering-benchmark

Nous allons nous limiter à l'études des jeux de données dits « artificiels ». Il s'agit de datasets 2D synthétiquement créés pour des fins pédagogiques.

1. Clustering k-means – TP 1

Dans cette partie du rapport, nous allons nous concentrer sur la technique k-means. Nous allons tout d'abord commencer pour l'introduire :

1.1 Rappel de K-means

L'algorithme KMeans regroupe les données en essayant de séparer les échantillons en n groupes de variance égale, en minimisant un critère connu sous le nom de somme des carrés d'inertie ou intra-groupe. Cet algorithme nécessite que le nombre de groupes soit spécifié. Il s'adapte bien à un grand nombre d'échantillons et a été utilisé dans un large éventail d'applications dans de nombreux domaines différents.

L'algorithme des k-means divise un ensemble de **N** échantillons **X** en **K** groupes disjoints **C**, chacun décrit par la moyenne $\boldsymbol{\mu}_j$ des échantillons de la grappe. Les moyennes sont communément appelées les "centroïdes" des clusters ; notez qu'elles ne sont pas, en général, des points de **X**, bien qu'elles vivent dans le même espace.

L'algorithme K-means vise à choisir des centroïdes qui minimisent l'inertie, ou le critère de la somme des carrés à l'intérieur du groupe :

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

L'inertie peut être considérée comme une mesure de la cohérence interne des clusters.

1.2 Coefficient Silhouette

Si les données ne sont pas labelisées (ç à d, que nous ne connaissons pas à l'avance quelle donnée appartient à quel cluster), l'évaluation doit être effectuée en utilisant le modèle lui-même. Le Coefficient de Silhouette est un exemple d'une telle évaluation, où un score plus élevé du Coefficient

de Silhouette correspond à un modèle avec des clusters mieux définis. Le coefficient de silhouette est défini pour chaque échantillon et se compose de deux scores :

a : La distance moyenne entre un échantillon et tous les autres points de la même classe.

b : La distance moyenne entre un échantillon et tous les autres points de la grappe la plus proche.

Le Coefficient de Silhouette s pour un seul échantillon est alors donné comme suit :

$$s = \frac{b - a}{max(a, b)}$$

1.3 Déroulement du TP

Pour ce premier TP, nous avons écris un script python qui réalise les opérations suivantes :

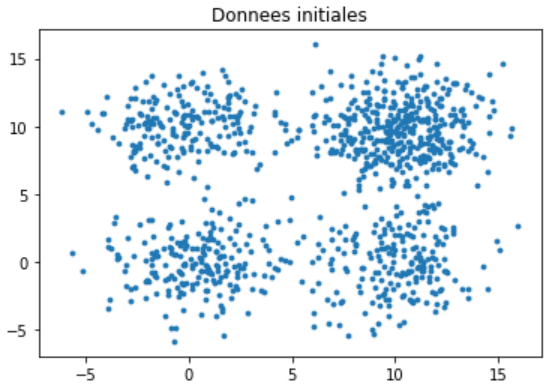
- Lecture et affichage des données initiales
- Appel de K-means pour une valeur de k fixée
- Recherche du k (nombre de clusters) optimal, à partir d'une liste de valeurs possible de nombre de clusters. Cette recherche est :
    - Visuelle en se basant sur la méthode du coude
    - Programmatique : En cherchant le k qui optimise le coefficient silhouette

Dans ce paragraphe, on vous montre deux exemples :

- Cas de réussite de K-means
- Cas d'échec de K-means

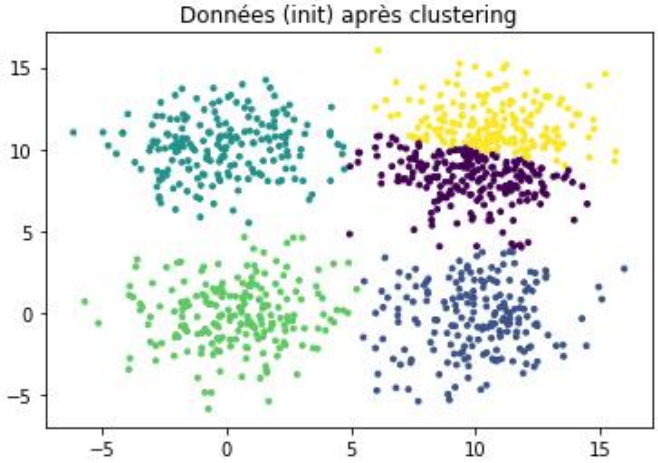
1.3.1 Cas de réussite de K-means

Ici, nous appliquons K-means sur le dataset « sizes1.arff »



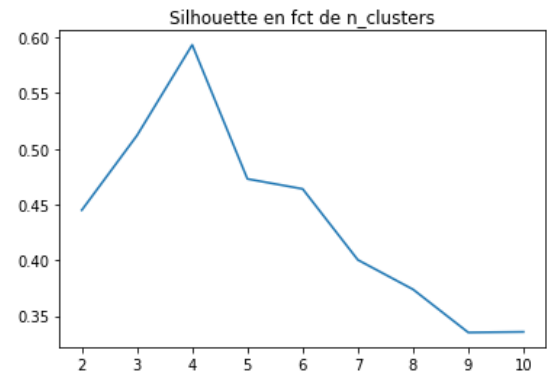
Nous partons d'un k fixée (k = 5) pour ensuite rechercher le k optimal

Yosra ZEYRI NEMRI – MS IoT
Lien GitHub : https://github.com/yosraZEYRINEMRI/MSIoT-Big-Data

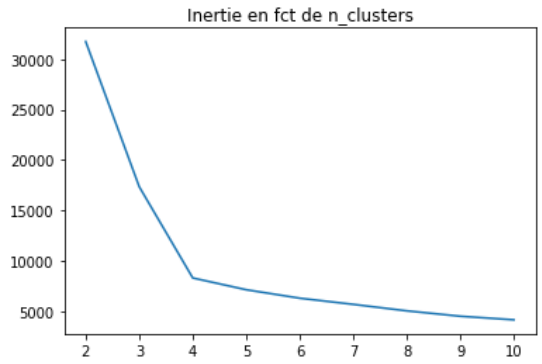


Après exploration de plusieurs valeurs de n_clusters possible (2 jusqu'à 10) , nous nous intéressons à l'évolution du coefficient de silhouette, l'inertie, la durée d'execution et le nombre d'iterations en fonction du paramètre k.

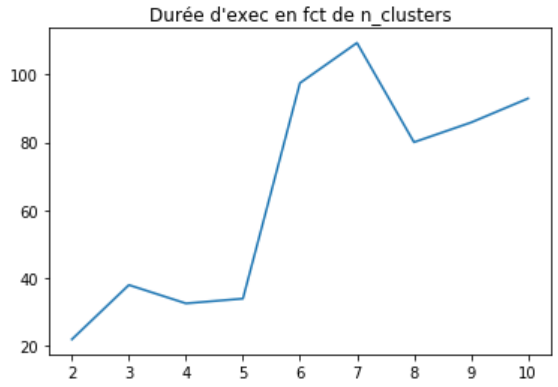
Silhouette en fct de #N_clusters



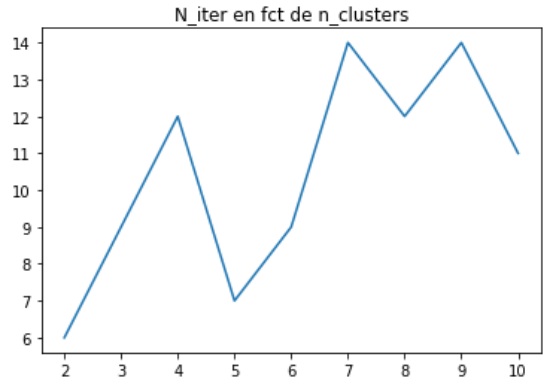
Inertie en fct de #N_clusters



Durée d'exec en fct de #N_clusters



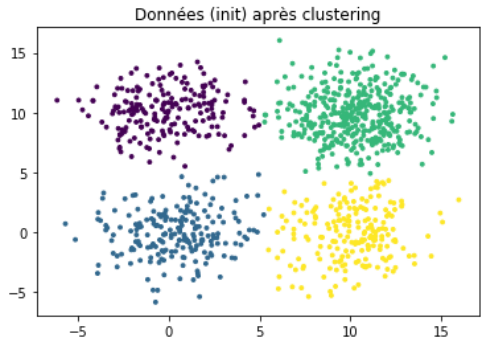
N_iter en fct de #N_clusters



Notre algorithme retient la valeur k = 4 comme valeur optimale. Il s'agit

- Du point de cassure (du coude) dans la courbe de l'inertie
- Il s'agit de la valeur qui maximise le coefficient de silhouette

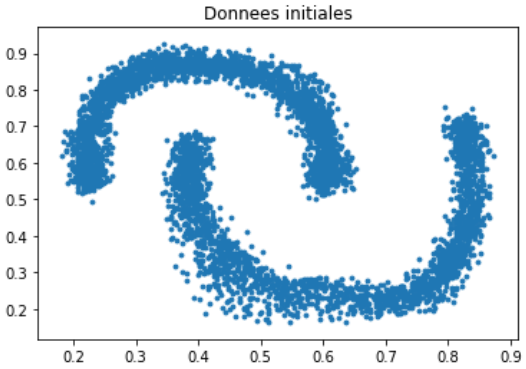
Après classification des données, nous obtenons la visualisation suivante du clustering optimal :



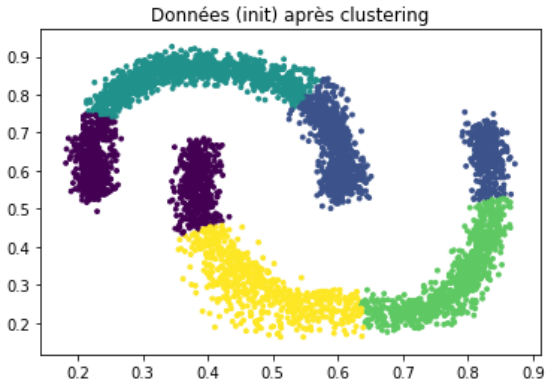

```
nb clusters = 4 , nb iter = 12 , runtime =  92.96 ms
Inertie :  8301.410417377861
Coefficient de silhouette :  0.5934089590658193
```

1.3.2 Cas d'échec de l'algorithme K-means

Ici, nous appliquons K-means sur le dataset « banana.arff »

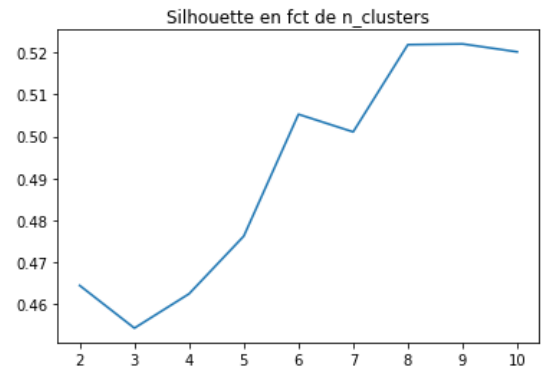


Nous partons d'un k fixée (k = 5) pour ensuite rechercher le k optimal

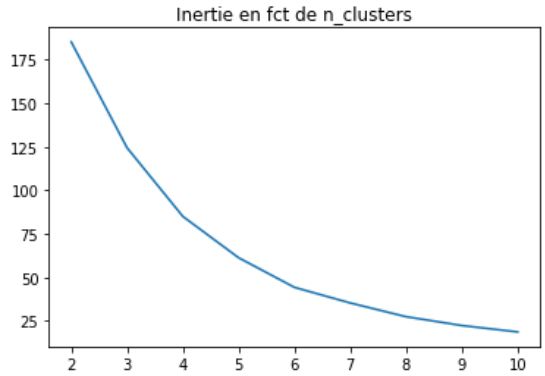


Après exploration de plusieurs valeurs de n_clusters possible (2 jusqu'à 10) , nous nous intéressons à l'évolution du coefficient de silhouette, l'inertie, la durée d'execution et le nombre d'iterations en fonction du paramètre k.

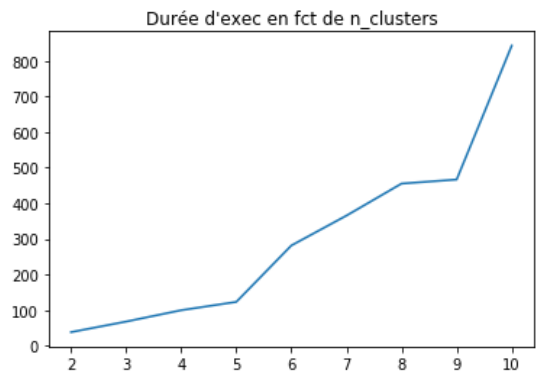
Silhouette en fct de #N_clusters



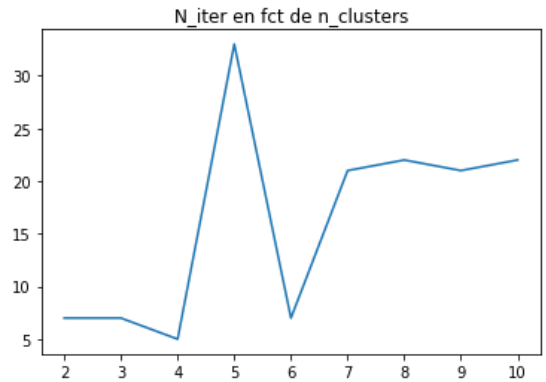
Inertie en fct de #N_clusters



Durée d'exec en fct de #N_clusters

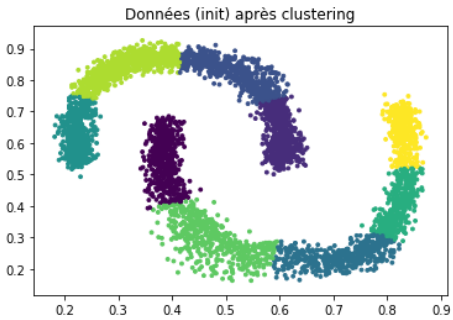


N_iter en fct de #N_clusters



Notre algorithme retient la valeur k = 9 comme valeur optimale. En réalité, on s'attendait à avoir k = 2. Ce cas d'usage démontre les limites de l'algorithmes K-means qu'on va discuter dans le paragraphe suivant :

Après classification des données, nous obtenons la visualisation suivante du clustering optimal :




```
nb clusters = 9 , nb iter = 21 , runtime =  842.53 ms
Inertie :  22.356631138361728
Coefficient de silhouette :  0.5220326139333544
```

1.4 Limites de l'algorithmes K-means

K-means souffre de divers inconvénients :

L'inertie fait l'hypothèse que les clusters sont convexes et isotropes (ce qui n'est pas le cas du dataset banana). Elle répond mal aux clusters allongés, ou aux manifolds de formes irrégulières.

En outre, l'inertie n'est pas une métrique normalisée : nous savons simplement que des valeurs plus faibles sont meilleures et que zéro est optimal. Mais dans les espaces de très haute dimension, les distances euclidiennes ont tendance à gonfler (c'est un exemple de ce qu'on appelle la "malédiction de la dimensionnalité"). L'exécution d'un algorithme de réduction de la dimensionnalité, tel que l'analyse en composantes principales (ACP), avant le clustering k-means peut atténuer ce problème et accélérer les calculs. (Nous n'avons pas bien sur eu affaire à des jeux de donnée à haute dimensions, le TP se limite à des cas d'usage 2D)

2. Clustering DBSCAN – TP 2

Dans cette partie du rapport, nous allons nous concentrer sur la technique k-means. Nous allons tout d'abord commencer pour l'introduire :

2.1 Rappel de la technique DBSCAN

L'algorithme DBSCAN considère les clusters comme des zones de haute densité séparées par des zones de faible densité. En raison de cette vision plutôt générique, les clusters trouvés par DBSCAN peuvent avoir n'importe quelle forme, contrairement aux k-means qui supposent que les clusters ont une forme convexe. L'élément central de DBSCAN est le concept d'échantillons centraux, qui sont des échantillons situés dans des zones de haute densité. Un cluster est donc un ensemble d'échantillons centraux, tous proches les uns des autres (mesurés par une certaine distance) et un ensemble d'échantillons non centraux qui sont proches d'un échantillon central (mais qui ne sont pas eux-mêmes des échantillons centraux). L'algorithme comporte deux paramètres, *min_samples* et *eps*, qui définissent formellement ce que nous entendons par "dense". Des *min_samples* plus élevés ou des *eps* plus faibles indiquent une plus grande densité nécessaire pour former un cluster.

Tout échantillon central fait partie d'une grappe, par définition. Tout échantillon qui n'est pas un échantillon central, et qui se trouve à une distance d'au moins *eps* de tout échantillon central, est considéré comme une aberration par l'algorithme.

Pour résumer :

- Le paramètre *min_samples* contrôle le degré de tolérance de l'algorithme vis-à-vis de bruit
- Le paramètre *eps* contrôle l'étendu du voisinage local des points

2.2 Déroulement du TP

2.2.1 Objectif et contributions

Pour cette partie, nous allons nous intéresser au jeu de données « banana » où on a eu un mauvais résultat de clustering avec la méthode K-means. Nous allons mettre en œuvre la méthode DBSCAN en essayant plusieurs valeurs possibles pour *min_samples* et *eps*.

Nous allons par la suite établir une approche systématique pour la recherche du paramètre *eps*.

Nous avons initialement un script qui va :

- Lire et afficher la donnée initiale
- Appeler DBSCAN avec le jeu de paramètre suivant :
    - Eps = 3 – min_samples = 5
    - Eps = 0.01 – min_samples = 5
    - Eps = 0.02 – min_samples = 5

Nous lui avons rajouter les parties suivantes :

- Evaluation du modèle grâce à la méthode silhouette

```
if len(set(cl_pred)) > 1:
    silh = metrics.silhouette_score(datanp, cl_pred, metric='euclidean')
    print("Coefficient de silhouette : ", silh)
```

- Automatisation de la recherche du eps optimal

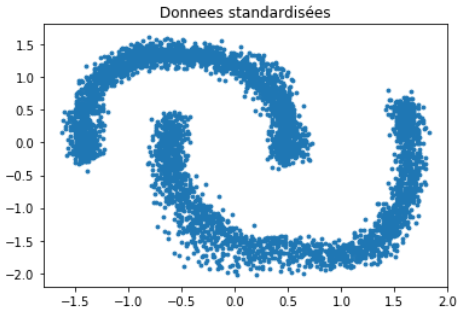
```
# Finding the optimal epsilon value

neigh = NearestNeighbors(n_neighbors=min_pts)
nbrs = neigh.fit(data_scaled)
distances, indices = nbrs.kneighbors(data_scaled)
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
plt.title("Nearest Neighbors : distance and associated indices")
plt.show()

kn = KneeLocator(distances, range(len(distances)), curve='convex', direction='increasing')
eps_opt = kn.knee
```

2.2.2 Résultats et analyse

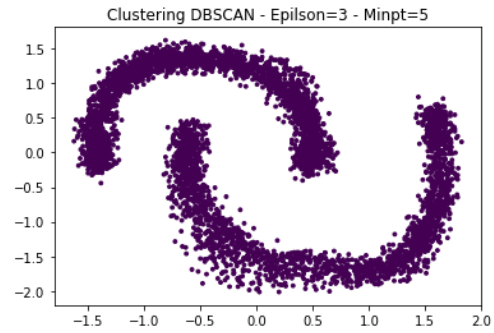
Pour rappel, initialement, la donnée a la distribution suivante :

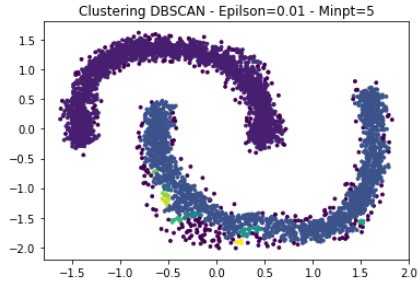


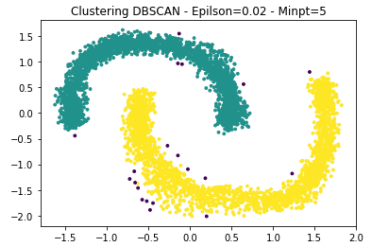
On n’est pas sur un nuage de point convexe et isotropique.

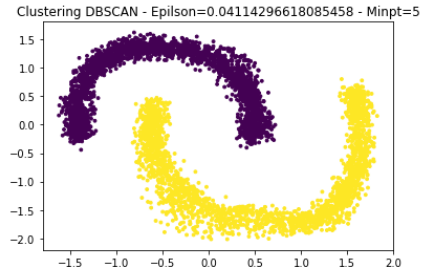
On trouve ci-dessous les résultats des expérimentations :

Yosra ZEYRI NEMRI – MS IoT
Lien GitHub : https://github.com/yosraZEYRINEMRI/MSIoT-Big-Data

<table>
<tr>
<td>Approche permissive
 

Estimated number of clusters: 1
Estimated number of noise points: 0</td>
<td>Approche trop conservatrice
 

Estimated number of clusters: 12
Estimated number of noise points: 284
Coefficient de silhouette : -0.3392992702277019</td>
</tr>
<tr>
<td>Approche « bon sens »
 

Estimated number of clusters: 2
Estimated number of noise points: 20
Coefficient de silhouette : 0.2323419449748864</td>
<td>Approche intelligente (systématique)
 

Estimated number of clusters: 2
Estimated number of noise points: 0
Coefficient de silhouette : 0.36733109590793084</td>
</tr>
</table>

Approche permissive

- Tous les points sont associés à un même cluster, 0 points considérés comme bruit
- Ceci est dû à un couple eps, min_samples à valeurs élevées (trop de tolérance au bruit)

Approche conservatrice

- 12 clusters au total, 284 points considérés comme bruit
- Ceci est dû à un eps trop faible (un gros cluster est donc vu comme plusieurs sous-clusters)

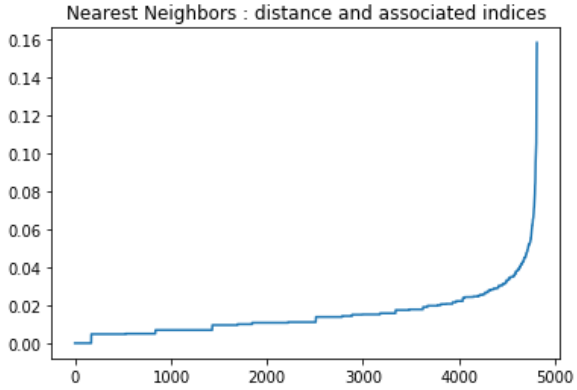
Approche bon sens

- 2 clusters au total, 20 points considérés comme bruit

Approche intelligente

- Dans cette approche, on systématise la recherche du eps optimal

- La valeur optimale correspond au « coude » dans la courbe des distances des nearest neighbors, pour ce jeu de donnée, celle-ci correspond à environ 0.04



- Résultat : 2 cluster + 0 bruit
- On remarque que c'est le jeu de paramètre qui maximise le coefficient de silhouette

3. Clustering agglomératif – TP 3

3.1 Famille des algorithmes de clustering agglomératif

Le clustering hiérarchique est une famille générale d'algorithmes de clustering qui construisent des clusters imbriqués en les fusionnant ou en les divisant successivement. Cette hiérarchie de clusters est représentée sous la forme d'un arbre (ou dendrogramme). La racine de l'arbre est le cluster unique qui regroupe tous les échantillons, les feuilles étant les clusters avec un seul échantillon.

L'objet AgglomerativeClustering effectue un clustering hiérarchique en utilisant une approche ascendante : chaque observation commence dans son propre cluster, et les clusters sont successivement fusionnés ensemble. Le critère de liaison détermine la métrique utilisée pour la stratégie de fusion :

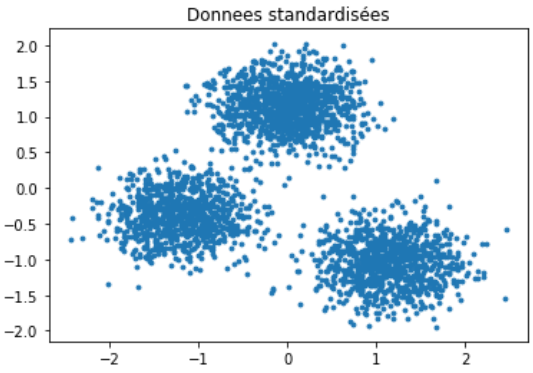
- **Ward** minimise la somme des différences au carré au sein de tous les clusters. Il s'agit d'une approche de minimisation de la variance et, en ce sens, elle est similaire à la fonction objectif de k-means mais abordée avec une approche hiérarchique agglomérative.
- Le **lien maximal** ou **complet** minimise la distance maximale entre les observations des paires de clusters.
- Le **couplage moyen** minimise la moyenne des distances entre toutes les observations de paires de clusters.
- Le **lien unique** minimise la distance entre les observations les plus proches des paires de clusters.

L'AgglomerativeClustering peut également s'adapter à un grand nombre d'échantillons lorsqu'il est utilisé conjointement avec une matrice de connectivité, mais il est coûteux en calcul lorsqu'aucune contrainte de connectivité n'est ajoutée entre les échantillons : il considère à chaque étape toutes les fusions possibles.

3.2 Déroulement du TP

3.2.1 Objectif et contributions

Nous avons repris le jeu de données XClara



Nous avons récupéré un script Python qui fait les opérations suivantes :

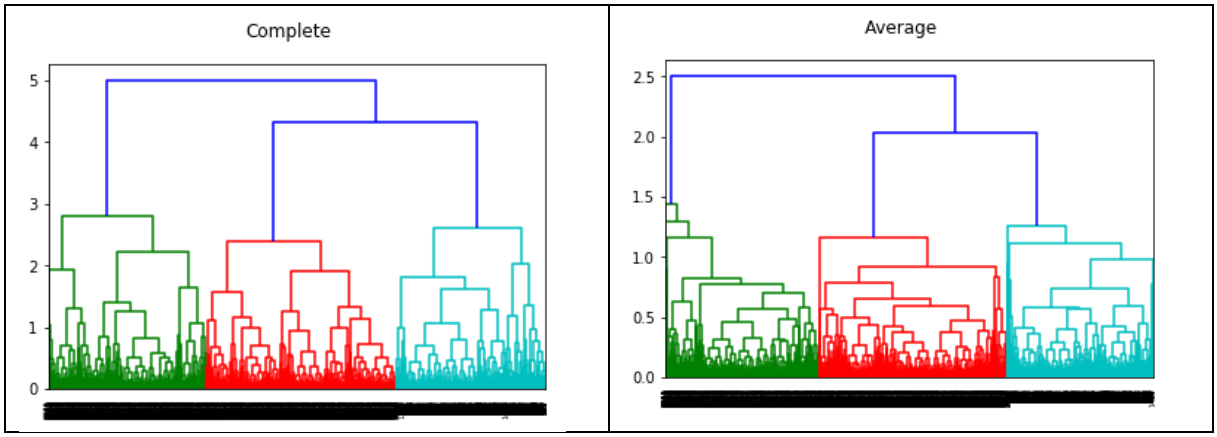
- Lecture et processing des données
- Visualisation du jeu de donnée initial
- Visualisation du dendrogrammes avec un linkage « complete »
- Réaliser le clustering pour k=3 et linkage = « complete » avec la méthode AgglomerativeClustering de Sklearn

Nous avons rajouté les contributions suivantes :

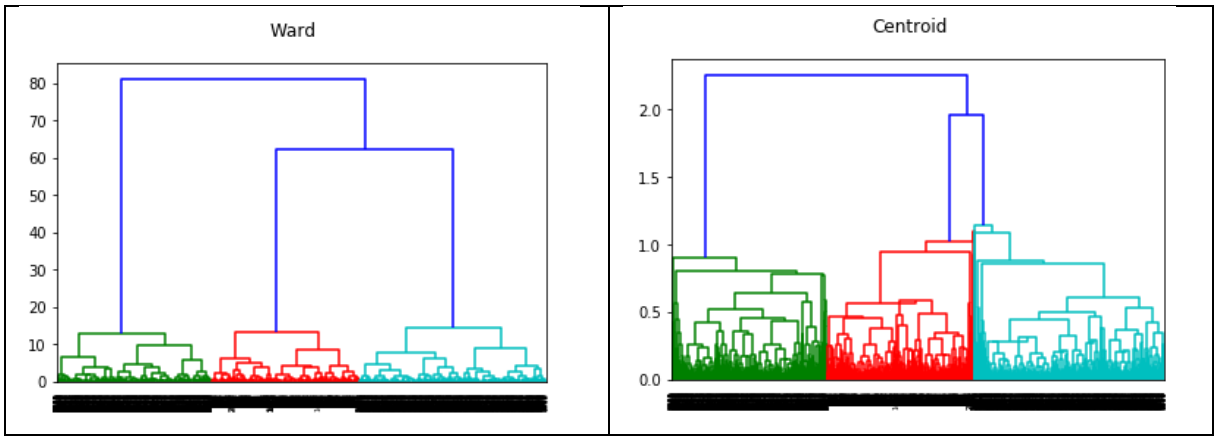
- Visualisation de dendrogrammes avec d'autres méthodes de linkage
- Evaluation avec la métrique Calinski Harabazs
- Evaluation avec la métrique Silhouette

3.2.2 Résultats et analyse

Différentes visualisations de dendrogrammes :

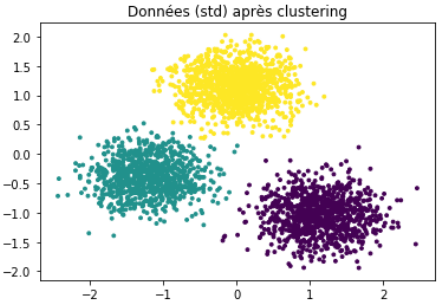


Yosra ZEYRI NEMRI – MS IoT
Lien GitHub : https://github.com/yosraZEYRINEMRI/MSIoT-Big-Data



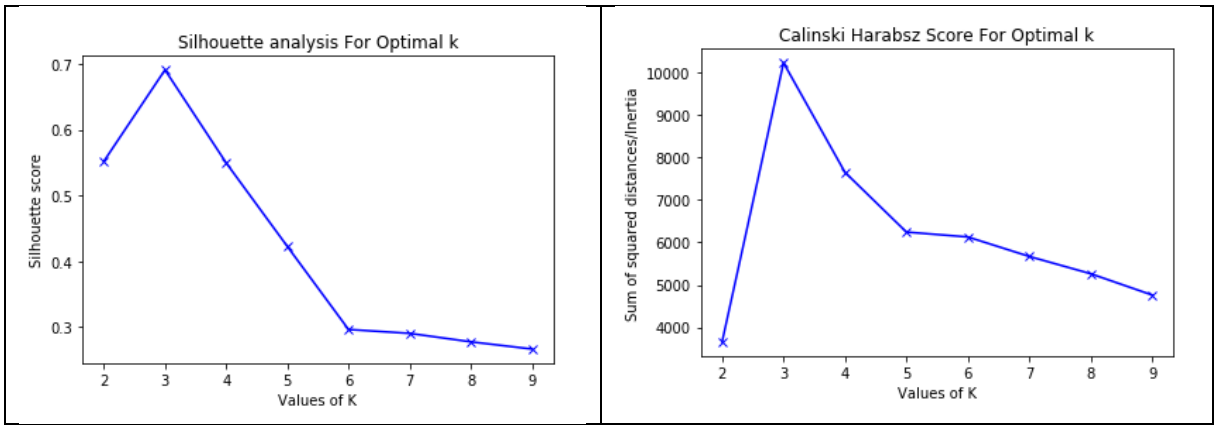
Tous les dendrogrammes donnent un nombre de clusters k = 3, et c'est ce qu'on va donner à AgglomerativeClustering comme paramètre.

On obtient le résultat suivant :




```
nb clusters = 3 , runtime =  436.39 ms
Coefficient de silhouette :  0.6908368385665083
Silhouette analysis - Elapsed time: 6.730581283569336 seconds
Calinski Harabsz score analysis - Elapsed time: 3.7545828819274902
seconds
```

Les routines d'évaluation de l'algorithme avec les métriques silhouette et Calinski donnent les résultats suivants :



Analyse :

- On arrive à un résultat satisfaisant

- Le temps d'exécution est 4 fois supérieur à celui de K-means
- L'affichage des dendrogrammes pour avoir une estimation de n_clusters prends nécessite beaucoup de temps de calcul

4. Test sur données réels

Nous allons dans cette partie tester l'algorithme K-means sur des données réelles. Nous allons nous intéresser en particulier au jeu de donnée « new-data/d32.txt ». Le code a toutefois été testé sur tous les autres fichiers.

Avec la méthode *df.shape* on peut connaitre le nombre d'échantillons et de colonnes figurant dans le jeu de données. Pour d32.txt, il s'agit :

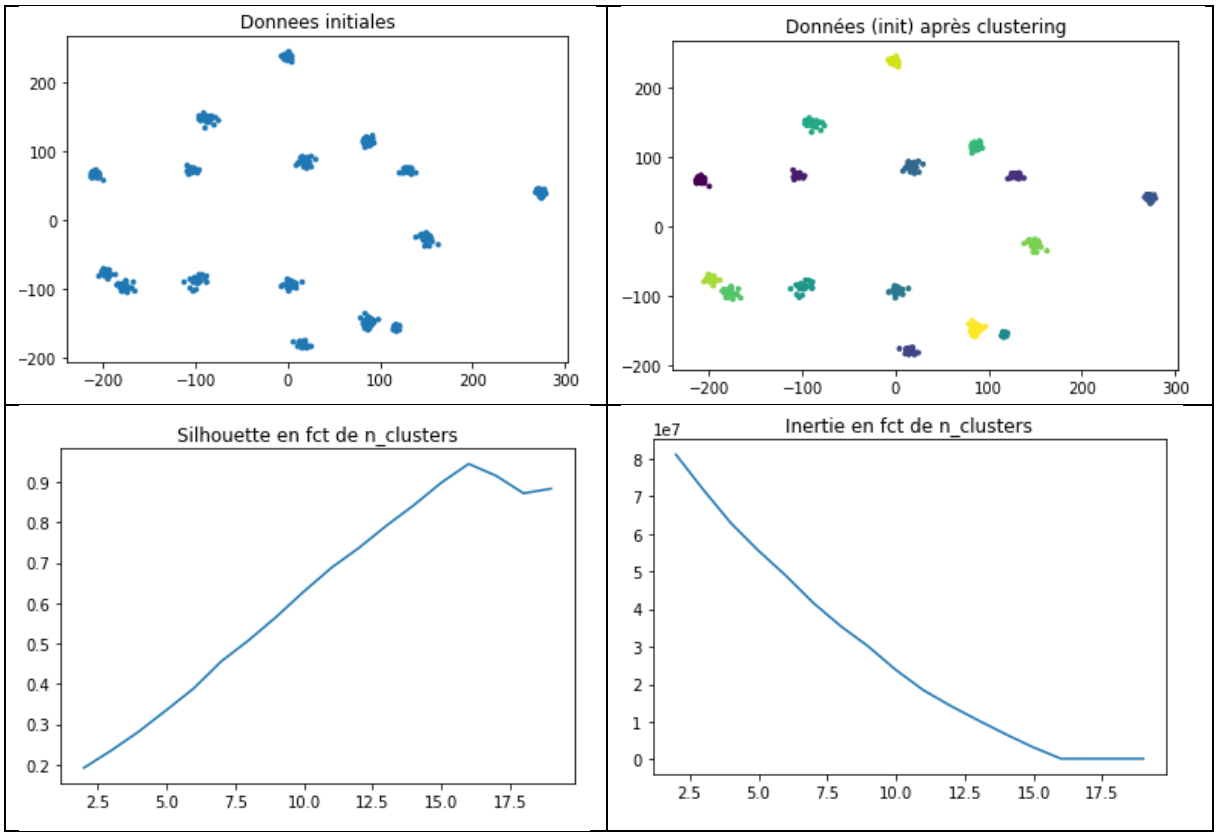
- De 1024 échantillon
- Et 32 colonnes (ou features)

Afin de pouvoir analyser visuellement le résultat du clustering, on va procéder à la réduction de la dimensionalité du problème en utilisant la technique d'Analyse Par Composantes Principales (PCA) :

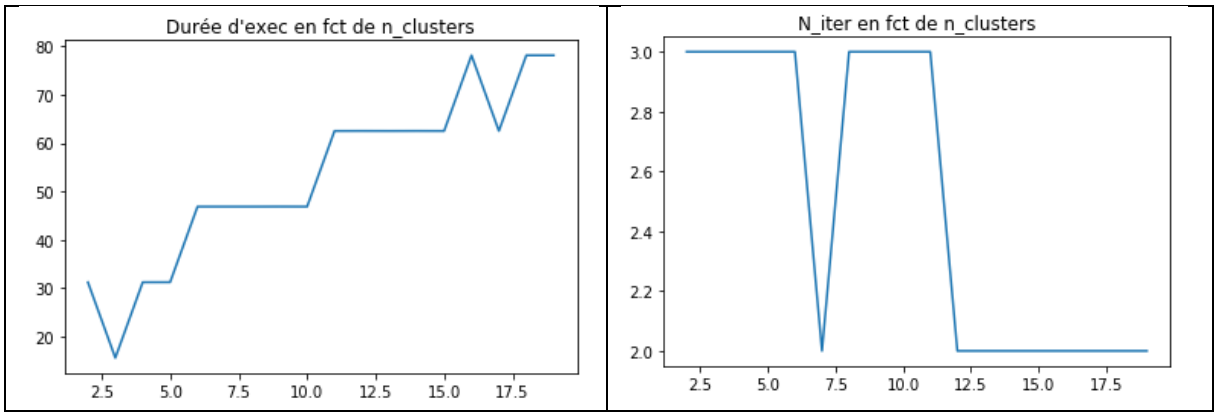
pca_transformer = PCA(n_components=2)

datanp_reduced = pca_transformer.fit_transform(datanp)

Voici les résultats :



Yosra ZEYRI NEMRI – MS IoT
Lien GitHub : https://github.com/yosraZEYRINEMRI/MSIoT-Big-Data



Notre algorithme retient la valeur k = 16 comme valeur optimale. Il s'agit

- Du point de cassure (du coude) dans la courbe de l'inertie
- Il s'agit de la valeur qui maximise le coefficient de silhouette

nb clusters = 16 , nb iter = 2 , runtime = 78.1 ms

Inertie : 232508.58655753967

Coefficient de silhouette : 0.9455303337060267

Conclusion

Dans cette série de TP, nous avons pu étudier, manipuler et analyser 3 familles d'algorithmes de Clustering :

- K-means
  - Avantage : Rapide
  - Inconvénient :
    - Suppose que la distribution des données est convexe et isotropique
    - Nécessite de deviner le nombre de clusters
- DBSCAN
  - Avantage : NbClusters est une sortie de l'algorithme et non une entrée
  - Inconvénient : Moins rapide que K-means
- Clustering Hiérarchique
  - Avantage :
    - Facilité de traitement de toute forme de similitude ou de distance.
    - Par conséquent, applicabilité à tous les types d'attributs.
  - Inconvénient : Durée d'exécution - Le clustering hiérarchique nécessite le calcul et le stockage d'une matrice de distance n×n. Pour les très grands ensembles de données, cela peut être coûteux et lent.