République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique
Spécialité : Big Data Analytics

Module : TIAD

Rapport du TP : Clustering

Travail présenté à Monsieur Necir

Réalisé par le binôme :

AISSANI Anouar

GOUMEZIANE Kenza

Année Universitaire : 2022/2023

Introduction :

Dans ce travail nous essayons de trouver une meilleure méthode de clustering qui aide à mettre chaque élément dans son cluster directement sans faire des calculs pour trouver le nombre de clusters.

Étapes de travail :

1) Importer les données :

* Dans notre cas on traite les deux colonnes (sepal length et sepal width) du data set (IRIS).

* IRIS : est un data set qui contient les données sur trois différents types de plantes (Setosa,Versicolour, et Virginica).

* Nombres de colonnes : 5
sepal length, sepal width, petal length, petal width, species

* Nombre de lignes : 150
50 lignes pour le type Setosa, 50 lignes pour le type Versicolour, 50 lignes pour le type Virginica

* L'ensemble de données ne contient aucune valeur “nulle”.
* Toutes les données ont le même type “float64”.

* On attribue les deux colonnes (sepal length et sepal width) dans le data frame qu'on va utiliser

X: sepal length
Y: sepal width

```
[ ]  iris = load_iris()
     iris.data
     d = iris.data[:,0:2]
     df = pd.DataFrame(data=d, columns=[['X','Y']])
     df
```

	X	Y
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
...	...	...
145	6.7	3.0
146	6.3	2.5
147	6.5	3.0

Figure 1.1 : les données utilisées

2) Normalisation des données :

* On utilise la fonction StandarScaler de la bibliothèque sklearn.preprocessing pour normaliser lesdonnées.
* Le score standard d'un échantillon x est calculé comme suit :

z = (x - m) / s
Où m est la moyenne des échantillons, et s est l'écart type des échantillons.

```
[ ]  scaler = StandardScaler()
     df[['X','Y']] = scaler.fit_transform(df[['X','Y']])
     df
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils
  FutureWarning,
```

	X	Y
0	-0.900681	1.019004
1	-1.143017	-0.131979
2	-1.385353	0.328414
3	-1.506521	0.098217
4	-1.021849	1.249201
...	...	...
145	1.038005	-0.131979
146	0.553333	-1.282963
147	0.795669	-0.131979

Figure 2.1 : normalisation des données

3) Appliquer l'algorithme :

Notre algorithme consiste à :

* Pour mettre chaque élément dans son cluster, il faut avoir au moins deux clusters au début, donc :
* On calculer la distance minimale entre tous les éléments de notre data set.
* On forme le premier cluster c1 à partir les deux éléments qui ont la distance minimale entre toutes les données.
* On calcule le centre du premier cluster cc1 et on cherche l'élément le plus éloigné de ce point p (pour créer le deuxième cluster c1).
* On créer un deuxième cluster c2 à partir de ce dernier point p.
*Maintenant, on a deux cluster c1 et c2.

* Pour le reste des éléments pi :

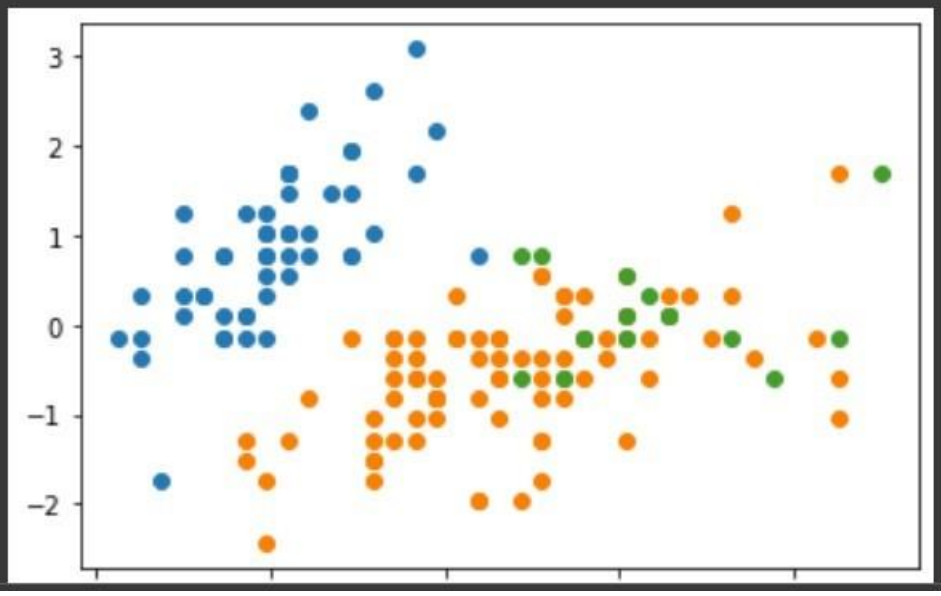
- On calcule le centre de chaque cluster cci= {cc1, cc2, ...ccn}
-On calcule le centre de tous les clusters k.
- On calcule la distance entre chaque élément et tous les centres des clusters.
- On prend le centre le plus proche de cet élément ci.

i. On compare la distance entre pi et le centre le plus proche d(pi, c) avec la moyenne des distances m , si d(pi,c)/m ≥ (minD +maxD)/2, alors on ajoute cet élément au plus proche cluster sinon on l'en crée un nouveau cluster ci.
ii. On répète l'algorithme jusqu'à la liste des points est vide.

4) Visualisation :

* Le résultat de notre algorithme :

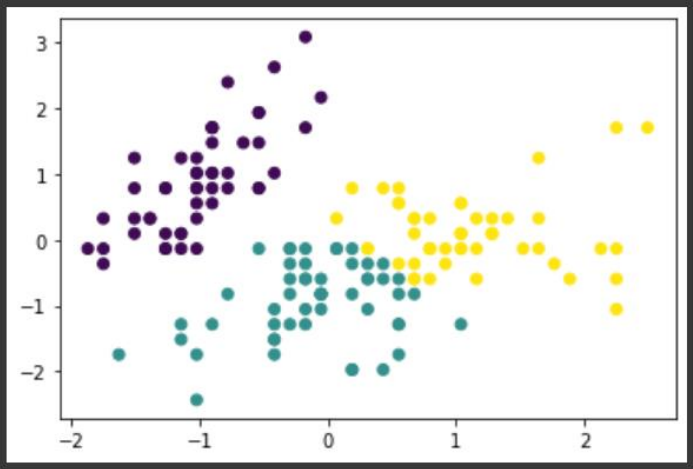
```
[ ]     c1 = []
        c2 = []
        j = 0
        while (j < len(c[i]-1)):
          c1.append(c[i][j])
          c2.append(c[i][j+1])
          j += 2
        clusterlist.append(c1)
        clusterlist.append(c2)
        plt.scatter(c1, c2)
```



Figures 4.1. Visualisation de notre algorithme

* Le résultat de Kmeans:

```
[ ]  kmeans = KMeans(n_clusters=3)
     kmeans.fit(df[['sepal length','sepal width']])
     df['kmeans_3'] = kmeans.labels_
     df
     plt.scatter(x=df['sepal length'],y = df['sepal width'], c = df['kmeans_3'])
     plt.show()
```

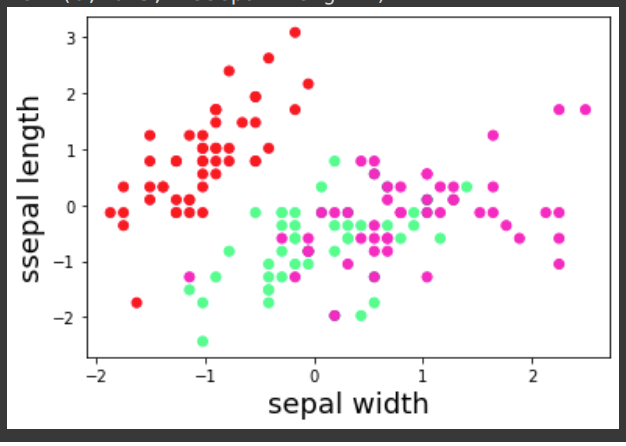


Figures 4.2. Visualisation de résultats Kmeans

* La classification depuis le dataset :

```
plt.scatter(x=df['sepal length'],y = df['sepal width'], c = iris.target, cmap='gist_rainbow')
plt.xlabel('sepal width', fontsize=18)
plt.ylabel('sepal length', fontsize=18)
```

```
Text(0, 0.5, 'ssepal length')
```



Figures 4.1. Visualisation de clustering du dataset

5) Comparaison entre notre résultat avec les résultats de KMEANS :

- Comparaison entre le nombre des individus de chaque groupe de notre algorithme et le nombre des individus de chaque groupe de Kmeans :

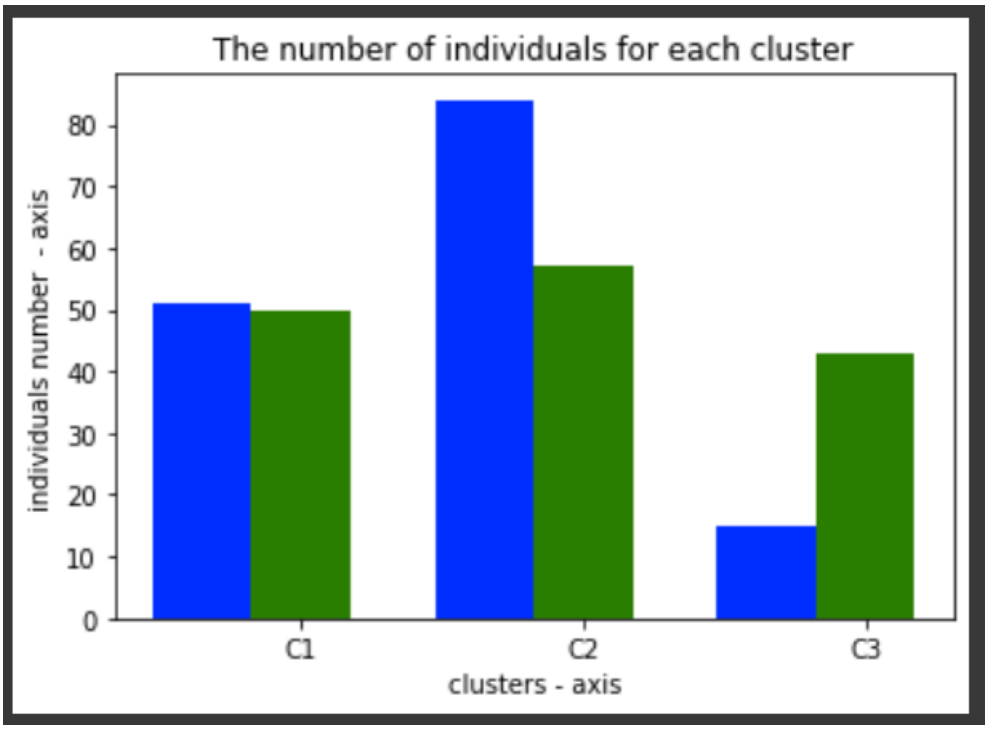


Figure 5.1 : le nombre des individus du chaque cluster pour les deux algorithmes

On remarque que :
* les premiers clusters dans les deux algorithmes ont presque le même nombre des individus.
* Pour les deuxièmes et les troisièmes clusters il y a une différence notable entre les deux algorithmes.

- Comparaison entre la distance inter de notre algorithme et Kmeans :

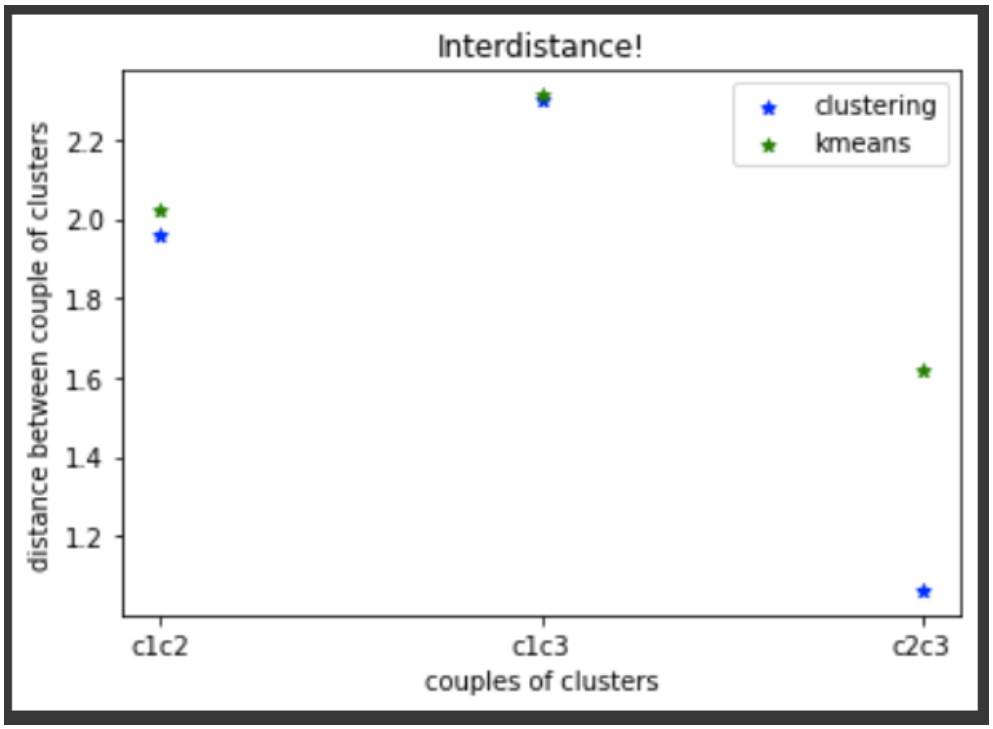


Figure 5.2 : Comparaison entre les distances inter des deux algorithmes

On remarque que :
* La distance inter entre les clusters c1, c2 et c1, c3 pour les deux algorithmes sont très proches, mais il y a une grande différence entre la distance inter entre c2, c3.

Conclusion :

* On a pu développer un algorithme qui affecte les individus dans leurs clusters directement.
* On a obtenu le nombre exact du cluster.
* Il y a une différence entre le nombre des individus de chaque cluster.
* les distances inter sans presque les mêmes.
* Notre algorithme doit être développé pour améliorer le nombre des individus de chaque cluster.