Atelier 3 : Clustering

Pr. Ali Idri

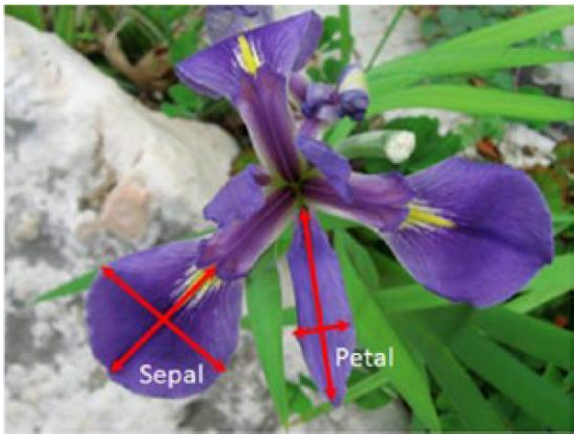
Master of Science in Quantitative and Financial Modelling

2022-2023

L'objectif de cet atelier est de prévoir les groupes (clusters) qui séparent le mieux la dataset Iris en utilisant le package scikit-learn et les deux algorithmes :

- K-means.
- Classification Ascendante Hiérarchique.

La dataset Iris est largement utilisée en apprentissage automatique et en analyse de données. Elle contient des mesures de la longueur du sépale, de la largeur du sépale, de la longueur du pétale et de la largeur du pétale de 150 fleurs d'iris, réparties en trois espèces différentes : Iris setosa, Iris versicolor et Iris virginica.



En utilisant les mesures de longueur et de largeur des sépales et des pétales, on peut appliquer des algorithmes de clustering pour regrouper les fleurs d'iris en fonction de leur similitude.

Le but est de trouver des groupes de fleurs similaires en fonction de leurs caractéristiques, ce qui peut aider à identifier des relations entre les différentes espèces d'iris et à mieux comprendre les différences entre elles.

Partie I : K-means

- **Rappel :**

K-Means (k-moyennes, centre mobiles) est un algorithme d'apprentissage non supervisé utilisé pour résoudre les problèmes de clustering. Il suit une procédure simple qui consiste à classer un ensemble de données dans un nombre de clusters, défini par la lettre « K », qui est fixé au préalable.

1- Importation des librairies :

On commence par importer les librairies nécessaires pour cette première partie de l'atelier.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn import metrics
```

- La bibliothèque Scikit-learn de Python destinée à l'apprentissage automatique approvisionne le module **sklearn.cluster** qui contient les méthodes de clustering. On charge depuis ce module, la classe **KMeans**.

2- Importation de la dataset :

Nous allons utiliser la base de données Iris implémenté par défaut sur sklearn. On utilise la méthode **load_iris()** pour charger la dataset Iris et puis on extrait les données dans la variable X:

```
# Chargement des données
iris = datasets.load_iris()
X = iris.data
```

3- Détermination de la valeur optimale de K :

-Nous savons au préalable que la dataset Iris est divisé en trois classes différentes : « Iris setosa », « Iris versicolor » et « Iris virginica ».

-Nous appliquerons sur cette dataset l'algorithme K-Means pour confirmer ou infirmer cette classification.

-Pour implémenter K-Means, on doit définir au préalable le nombre de clusters K. Le choix du nombre de clusters K n'est pas forcément intuitif, le but est de trouver un K optimal.

-L'une des méthodes les plus populaires pour y arriver est la méthode d'Elbow (la méthode du coude). Cette méthode consiste à exécuter K-Means avec différentes valeurs de K et de calculer la variance des différents clusters.

Pour une gamme de clusters K (de 1 à 10), on implémente la méthode K-Means pour les différentes valeurs de K et pour chaque valeur, nous calculons l'inertie intra-cluster (WCSS : Within-Cluster Sums of Squares) définit comme suit : La somme des distances euclidiennes entre chaque point et son centroïde associé.

L'attribut « **kmeans.inertia_** » retourne la valeur finale d'inertie intra-cluster de la méthode K-Means.

```
tab=[]
for i in range(1,10):
    kmeans=KMeans(n_clusters=i)
    kmeans.fit(X)
    tab.append(kmeans.inertia_)
tab
```

- On sauvegarde la valeur de l'inertie intra-cluster pour chaque valeur de K dans un tableau, qu'on affiche par la suite :

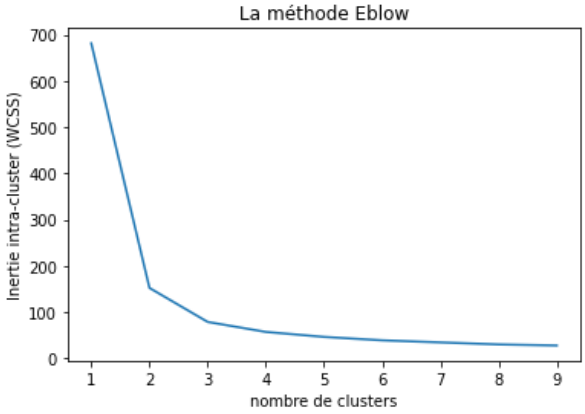
```
[681.3706,
 152.3479517603579,
 78.851441426146,
 57.22847321428572,
 46.446182051282065,
 39.054977867477874,
 34.574433214128874,
 30.1865551948052,
 27.89549464570518]
```

4- Visualisation de la méthode Elbow :

-On place les valeurs d'inertie intra-cluster obtenues en fonction du nombre de clusters sur un graphique qu'on visualise.

-Le point du coude est celui du nombre de clusters à partir duquel l'intertie intra-cluster ne se réduit plus significativement. Le fait de chercher le point représentant le coude, a donné nom à cette méthode : La méthode Elbow.

```
plt.plot(range(1,10),tab)
plt.title("La méthode Eblow")
plt.xlabel("nombre de clusters")
plt.ylabel("Inertie intra-cluster (WCSS)")
plt.show()
```



À travers le graphique ci-dessus, nous pouvons observer que le point de retournement de cette courbe est à la valeur de 3. Selon la méthode d'Elbow, la valeur optimale de K est 3. Ce qui confirme les classes initiales définies pour cette dataset.

5-Visualisation des clusters :

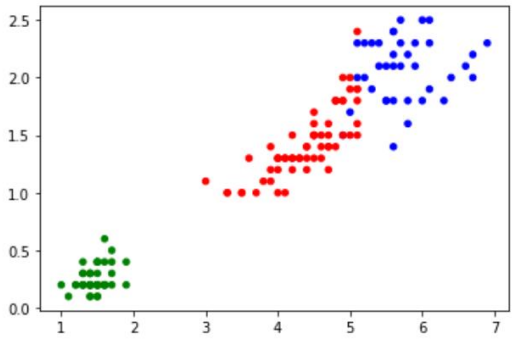
-On implémente l'algorithme de K-Means avec K=3, et on visualise graphiquement les clusters formés pour la dataset Iris en fonction de la longueur et largeur des pétales.

- Les groupes associés aux exemples d'apprentissage sont stockés dans l'attribut « **kmeans.labels_** ».

- L'attribut « **kmeans.cluster_centers_** » retourne les cordonnées des centres des clusters.

```
#Application deK-Means avec 3 clusters
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
#Visualisation
colormap=np.array(["red", "green", "blue"])
plt.scatter(X[:,2], X[:,3], c=colormap[kmeans.labels_], s=20)
```

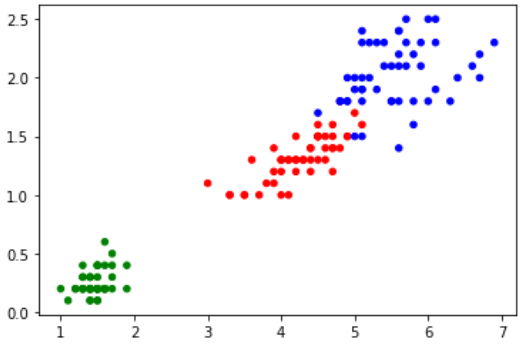
```
plt.show()
```



-Le graphique montre bien 3 classes d'observations respectivement en vert, rouge et bleu.

-On visualise les données de la dataset Iris en fonction de leurs classes dans un graphique et on le compare avec le graphique précédent pour voir si les 3 clusters correspondent aux 3 classes :

```
colormap=np.array([ "green","red","blue"])
plt.scatter(X[:,2], X[:,3], c=colormap[iris.target], s=20)
plt.show()
```



Question : -Visualisez les clusters prédits par K-Means en fonction de la longueur et largeur des sépales ainsi que leurs centres.

6- Modification des paramètres par défaut :

-On applique la classification automatique avec K-means, en modifiant les paramètres par défaut :

- **init** : La méthode d'initiation (aléatoire ou avec la technique k-means++)
- **n_init** : le nombre de fois où le cluster K-means va initier les centres des clusters.

- Il est possible d'évaluer la cohérence entre les classes de départ et le partitionnement trouvé par K-Means en utilisant l'indice de Rand ajusté. L'appel à **metrics.adjusted_rand_score()** compare le partitionnement obtenu par la classification automatique (kmeans.predict) avec le partitionnement correspondant aux groupes définis au départ (iris.target).

On implémente K-Means avec n_init = 100 et on calcule la cohérence :

```
kmeans = KMeans(n_clusters=3, n_init=100, init='k-means++').fit(X)
metrics.adjusted_rand_score(kmeans.predict(X),iris.target)
```

Travail demandé :

1-Implémentez K-Means avec n_init = 1 et init = « k-means++ » 10 fois et calculez l'indice de Rand ajusté.

2-Implémentez K-Means avec n_init = 1 et init = « random » 10 fois et calculez l'indice de Rand ajusté.

3- Complétez le tableau suivant. Que constatez-vous ?

K-means++	1	2	3	4	5	6	7	8	9	10
L'indice de Rand ajusté										
random	1	2	3	4	5	6	7	8	9	10
L'indice de Rand ajusté										

4- Refaire les étapes précédentes avec n_init = 10.

Partie II : HAC (Hierarchical Ascendant Classification)

Rappel :

- Le principe de la classification ascendante est de regrouper (ou d'agréger), à chaque itération, les données et/ou les groupes les plus proches qui n'ont pas encore été regroupé(e)s. A partir de la hiérarchie de groupes résultante il est possible d'observer l'ordre des agrégations de groupes, d'examiner les rapports des similarités entre groupes (clusters), ainsi que d'obtenir plusieurs partitionnements à des niveaux de granularité différents.

- Un **dendrogramme** est un diagramme utilisé pour illustrer la hiérarchie des clusters générés par l'algorithme de HAC. Il s'agit de regrouper itérativement les individus, en commençant par le bas (les deux plus proches) et en construisant progressivement un arbre regroupant finalement tous les individus en une seule classe, à la racine.

1- Importation des librairies :

On commence par importer les librairies nécessaires pour cette deuxième partie de l'atelier.

```
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
```

Depuis la librairie scikit-learn, on charge depuis le sous module « cluster » l'algorithme HAC qu'on peut trouver avec la classe « **AgglomerativeClustering »**.

Depuis la librairie scipy, on charge depuis le module « cluster », le sous module « hierarchy » pour importer la classe « **dendtrogram** » qui permet de visualiser le dendrogramme de l'algorithme HAC.

2- Implémentation de HAC :

-Nous appliquerons cette fois-ci sur la dataset Iris l'algorithme HAC avec AgglomerativeClustering.

(On suppose que la partie I.2 de l'importation de la dataset est déjà faite)

-Les groupes associés aux exemples d'apprentissage sont stockés dans l'attribut « **labels_** ».

```
hac = AgglomerativeClustering()
hac = hac.fit(X)
print(hac.labels_)
```

-Quel est le nombre de clusters obtenus ? Pourquoi ?

3- Visualisation du dendrogramme :

La documentation officielle de sklearn a défini la fonction ci-dessous pour pouvoir visualiser le dendrogramme de la classe AgglomerativeClustering.

- L'attribut « **AgglomerativeClustering.children_** » retourne les fils de tous les nœuds.

- L'attribut « **AgglomerativeClustering. distances_** » retourne la distance entre les nœuds.

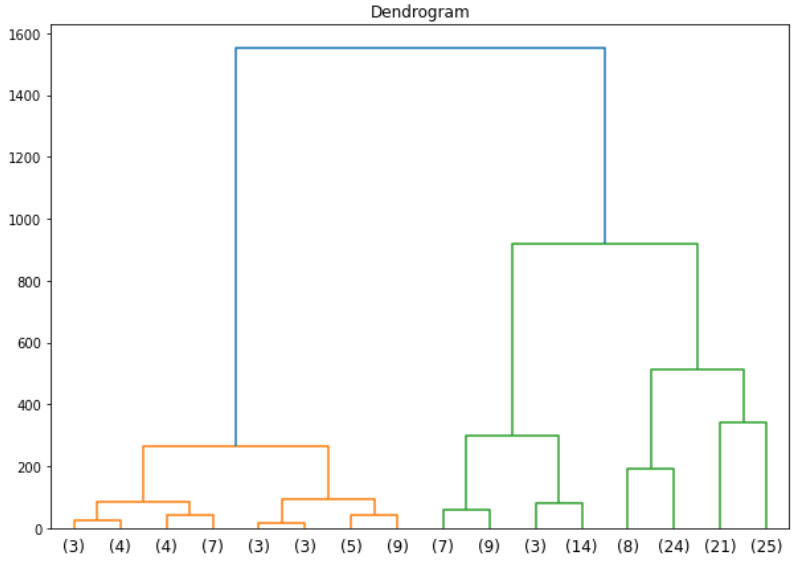
En clustering hiérarchique agglomératif (HAC), le "linkage" est une mesure de distance utilisée pour déterminer la proximité entre les clusters dans le processus de fusion. Plusieurs méthodes de linkage peuvent être utilisées en HAC, dont les plus courantes sont :

- Linkage simple : La distance entre deux clusters est déterminée par la distance minimale entre les points de chaque cluster.
- Linkage complet : La distance entre deux clusters est déterminée par la distance maximale entre les points de chaque cluster.
- Linkage moyenne : La distance entre deux clusters est déterminée par la moyenne des distances entre tous les points des deux clusters.
- Ward linkage : Cette méthode de linkage vise à minimiser la variance intra-cluster et à maximiser la variance inter-cluster en minimisant la somme des carrés des écarts à la moyenne.Pour pouvoir afficher

```
# Plotting the dendrogram
plt.figure(figsize=(10, 7))
plt.title("Dendrogram")
linkage_matrix = linkage(hac.children_, 'ward')
dendrogram = sch.dendrogram(linkage_matrix)
plt.show()
```

-Pour améliorer l'affichage du dendrogramme, c'est possible de définir d'autres paramètres pour la fonction « **plot_dendrogram** ». Le paramètre « **truncate_mode** » est utilisé pour rendre le dendrogramme plus lisible, on utilise le mode '**level**' qui limite le nombre de niveaux du dendrogramme qui seront affichés. Ce nombre est définit par le paramètre « **p** ».

```
hac = AgglomerativeClustering(n_clusters=3)
hac.fit(X)
# Plotting the dendrogram
plt.figure(figsize=(10, 7))
plt.title("Dendrogram")
linkage_matrix = linkage(hac.children_, 'ward')
dendrogram = sch.dendrogram(linkage_matrix,truncate_mode='level', p=3)
plt.show()
```



4- Modification des paramètres par défaut :

-On applique la classification automatique avec HAC, en modifiant les paramètres par défaut :

- **n_clusters** : Le nombre de groupes à former ainsi que le nombre de centroïdes à générer.
- **Linkage** : Détermine la méthode utilisée pour calculer la distance entre les groupes : ward, complete, average, ou single. (Par défaut c'est ward)
- **Affinity** : La métrique utilisée pour calculer 'Linkage' : euclidean, l1, l2, manhattan, cosine, ou precomputed. (Par défaut c'est euclidean)

Travail demandé :

1- Implémentez HAC avec n_clusters = 3 et complétez le tableau suivant. Que constatez-vous ?

Linkage	ward	complete	average	single	complete	average	single	complete	average	single
Affinity	euclidean	euclidean	euclidean	euclidean	manhattan	manhattan	manhattan	l1	l1	l1
L'indice de Rand ajusté										

2- Visualisez le dendrogramme pour linkage= « single » et linkage= « complete » en gardant les autres paramètres par défaut.