Université Moulay Ismail
École Nationale Supérieure D'arts Et Métiers
A.U : 2024/2025

TP2 – Clustering avec la méthode K-Means

L'algorithme K-Means est une méthode de clustering non supervisée utilisée pour regrouper des données en K groupes (ou clusters), selon leur similarité. Il fonctionne en alternant entre deux étapes :

1. **Assignation des points aux centroïdes les plus proches.**
2. **Mise à jour des centroïdes en calculant la moyenne des points de chaque groupe**

En Python, l'implémentation de K-Means est disponible via la classe **KMeans** du module **sklearn.cluster.**

```
KMeans(n_clusters=8, init='k-means++', n_init='auto', max_iter=300, tol=0.0001, verbose=0,
random_state=None, copy_x=True, algorithm='lloyd')
```

❖ Paramètres de KMeans()

Paramètre	Description
n_clusters	(int) Nombre de clusters à former.
init	{'k-means++', 'random'} ou ndarray – Méthode d'initialisation des centroïdes. (valeur par défaut : k-means++)
n_init	(int ou 'auto') Nombre d'exécutions indépendantes avec différentes initialisations.
max_iter	(int) Nombre maximal d'itérations pour une exécution.
tol	(float) Seuil de tolérance pour la convergence (valeur par défaut : 1e-4).
verbose	(int) Niveau de verbosité. 0 = silencieux. (Affiche des messages pendant l'exécution)
random_state	(int ou None) Contrôle la reproductibilité des résultats.
copy_x	(bool) Si True, copie les données avant le clustering.
algorithm	{'lloyd', 'elkan'} – Algorithme utilisé : lloyd (standard) ou elkan (plus rapide pour données denses).

❖ Attributs de KMeans()

Attribut	Contenu	À quoi ça sert
labels_	Groupe de chaque point	Pour colorer ou analyser les clusters
cluster_centers_	Coordonnées des centres	Pour analyser la position des clusters
inertia_	Somme des distances intra-cluster	Pour la méthode du coude
n_iter_	Nombre d'itérations	Pour analyser la convergence
n_features_in_	Nombre de variables en entrée	Vérification technique
feature_names_in_	Noms des colonnes (si DataFrame)	Pour garder une trace des variables

❖ Méthodes essentielles de KMeans()

Méthode	Description
fit(X)	Entraîne le modèle sur les données X.
predict(X)	Donne le cluster de chaque point de X (après entraînement).
fit_predict(X)	Combine fit et predict, retourne directement les étiquettes.
transform(X)	Calcule les distances entre chaque point de X et les centroïdes.
fit_transform(X)	Entraîne le modèle et retourne les distances aux centroïdes.

Remarque : une description complète, accompagnée d'exemples, est disponible dans la documentation officielle de Python : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Exercice 1

Dans cet exercice, nous utilisons la fonction **make_blobs** de la bibliothèque **sklearn.datasets** pour **générer un jeu de données simulé (synthétique ou artificielle)**. Cette fonction est très pratique pour créer rapidement des ensembles de points groupés autour de centres définis, ce qui est idéal pour tester des algorithmes de classification non supervisée comme **K-Means**.

Elle permet de :

- Définir le **nombre d'échantillons** (n_samples)
- Spécifier le **nombre de groupes (ou centres)** (centers)
- Contrôler la **dispersion des points autour de chaque centre** (cluster_std)
- Reproduire les mêmes résultats avec un **aléa contrôlé** (random_state)

Cela permet de visualiser facilement des clusters bien séparés, et de comprendre comment l'algorithme les identifie.

```
from sklearn.datasets import make_blobs
x, y = make_blobs(
    n_samples=300,              # Nombre total de points à générer
    centers=4,                  # Nombre de clusters
    n_features=2,               # Nombre de dimensions (par défaut = 2)
    cluster_std=1.0,            # Écart-type des clusters (plus petit = plus compact)
    center_box=(-10.0, 10.0), # Intervalle pour générer les centres (par défaut)
    shuffle=True,               # Mélange les points générés
    random_state=42             # Graine aléatoire pour reproductibilité
)
```

- X contient les coordonnées des points (array),
- y contient les **étiquettes réelles** des clusters (utile pour visualiser mais pas utilisé par K-Means).

Travail à faire

1) Importer les packages nécessaires pour générer les données synthétiques et pour appliquer l'algorithme K-Means.
2) Générer un jeu de données avec 600 observations, caractérisées par 2 attributs, répartir sur 4 groupes. Stocker les coordonnées dans la variable X et les classes dans Y.
3) Visualiser les coordonnées de X ainsi que les classes de chaque observation.
4) Afficher les données brutes dans un nuage de points. Les données sont-elles bien réparties ?

5) Appliquer l'algorithme K-Means (avec sklearn) pour K = 3.
6) Afficher les clusters obtenus ainsi que leurs centroïdes.
7) Utiliser la méthode du coude (Elbow method) pour estimer la valeur optimale de K.
8) Tester K-Means avec K = 4. Les clusters sont-ils mieux définis ?
9) Expliquer : pourquoi les centroïdes changent-ils à chaque exécution si l'on ne fixe pas le paramètre random_state ?

Exercice 2

L'objectif de cet exercice est d'appliquer l'algorithme K-Means pour identifier des groupes de fleurs similaires dans le jeu de données Iris, sans utiliser les étiquettes (species). Le jeu de données Iris contient 150 fleurs réparties en 3 espèces :

- Iris-setosa
- Iris-versicolor
- Iris-virginica

Chaque fleur est décrite par 4 variables :

- Longueur du sépale (sepal length)
- Largeur du sépale (sepal width)
- Longueur du pétale (petal length)
- Largeur du pétale (petal width)

1) Charger le jeu de données Iris depuis sklearn.datasets. (**from sklearn.datasets import load_iris ; X, y = load_iris(return_X_y=True)**)
2) Visualiser les données en 2D en testant toutes les combinaisons possibles de paires de variables, puis identifier celle qui offre la meilleure séparation visuelle entre les classes. Cette combinaison sera retenue comme support principal pour les visualisations des analyses à venir.

```
import seaborn as sns
import pandas as pd
df = pd.DataFrame(X, columns=load_iris().feature_names)
df["target"] = y
sns.pairplot(df, hue="target", palette={0: "orange", 1: "skyblue", 2: "green"})
```

3) Appliquer K-Means avec k = 3.
4) Déterminer le nombre d'itérations effectuées.
5) Afficher les clusters et centroïdes obtenus sur un graphique.
6) Comparer les clusters avec les vraies classes (matrice de confusion).
7) Évaluer le clustering à l'aide de métriques telles que NMI (Normalized Mutual Information), ARI (Adjusted Rand Index) et la mesure de silhouette.
8) Utiliser la méthode du coude pour estimer la valeur optimale de k.
9) Conserver les résultats précédents, puis appliquer à nouveau K-Means en définissant le paramètre init='random'.
10) Comparer le nombre d'itérations entre les deux exécutions, puis en tirer une conclusion.

Exercice 3

1) Chargement et exploration des données
   (a) Importez le dataset **load_wine** de **sklearn.datasets**.
   (b) Affichez la taille du jeu de données et la description des variables.
2) Appliquez l'algorithme K-Means avec 3 clusters dans les situations suivantes :
   (a) Sur les données brutes (non normalisées).
   (b) Sur les données normalisées avec StandardScaler.
   (c) Sur les 2 premières composantes principales (PCA) des données brutes.
   (d) Sur les 2 premières composantes principales (PCA) des données normalisées.
   Remarque : Pour (c) et (d), utilisez PCA(n_components=2).
3) Pour chaque cas (a → d), calculez les métriques suivantes :
   (a) Adjusted Rand Index (ARI) : adjusted_rand_score()
   (b) Normalized Mutual Information (NMI) : normalized_mutual_info_score()
   (c) Silhouette Score : silhouette_score()
   (d) Pour chaque cas, reportez ARI, NMI et Silhouette sous forme d'un tableau.
   (e) Dans quel cas obtient-on le meilleur clustering ? Expliquez pourquoi.
4) Visualisation:
   a) Représentez les résultats des cas (c) et (d) (PCA en 2D) avec un nuage de points coloré selon les labels prédits par K-Means.
   b) Que remarquez-vous visuellement dans la séparation des clusters avant et après normalisation dans l'espace PCA ?
5) Pourquoi est-il important de normaliser les données avant d'appliquer K-Means ?
6) PCA améliore-t-il la qualité du clustering ici ? Justifiez avec les scores.
7) Comparez le temps d'exécution des deux cas qui donnent le meilleur clustering. Lequel est le plus rapide ? Pourquoi ?
8) Essayez d'appliquer d'autres méthodes de normalisation que StandardScaler (par exemple MinMaxScaler, RobustScaler, Normalizer). Puis comparez les résultats obtenus (ARI, NMI, Silhouette) à ceux obtenus avec StandardScaler. Quelle méthode semble la plus adaptée dans ce contexte ?

Exercice 4

Appliquez l'algorithme **K-Means** aux jeux de données Fromage et Titanic. Essayez de déterminer la configuration permettant d'obtenir le meilleur regroupement possible des données, en vous appuyant sur les étapes et méthodes vues précédemment.

Exercice 5

Implémentez vous-même l'algorithme **K-Means** en Python, sans utiliser **sklearn.cluster.KMeans**. Votre implémentation doit respecter les étapes fondamentales de l'algorithme vues en cours.