Chapitre 5 : Apprentissage non supervisé

1. Notion de clustering.
2. Algorithmes :
    - **K-means**
    - DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
3. Visualisation 2D et interprétation des résultats.
4. **Exercices :**
    - Expliquer comment utiliser un algorithme de clustering sur un Dataset
    - Expliquer comment visualiser les clusters.
    - ….

Ce chapitre porte sur l'**apprentissage non supervisé**. Contrairement aux chapitres précédents, nous n'avons plus d'étiquettes (y) pour guider le modèle. L'IA doit découvrir par elle-même la structure cachée des données.

Chapitre 5 : Apprentissage non supervisé – Le Clustering

1. Notion de Clustering

Le clustering (ou partitionnement) consiste à regrouper des objets similaires dans des groupes appelés **clusters**, de telle sorte que :

- Les objets d'un même groupe soient très proches les uns des autres.
- Les objets de groupes différents soient les plus éloignés possibles.

Applications : Segmentation de clients, regroupement de gènes similaires, ou identification de régimes de fonctionnement dans une machine (ex: stable, transitoire, critique).

2. Algorithmes de référence

A. K-means (Moyennes mobiles)

C'est l'algorithme le plus populaire. Il fonctionne par itérations :

1. On choisit K (le nombre de clusters souhaités).
2. On place K centres (centroïdes) au hasard.
3. On affecte chaque point au centroïde le plus proche.
4. On déplace les centroïdes au centre des nouveaux groupes.
5. On recommence jusqu'à ce que les centres ne bougent plus.

- **Point faible :** Il faut connaître K à l'avance et il ne fonctionne bien que pour des groupes de forme circulaire/sphérique.

B. DBSCAN

Contrairement au K-means, DBSCAN se base sur la **densité**.

1. Il cherche des zones où les points sont très serrés.
2. Il peut découvrir des formes complexes (croissants, anneaux).
3. Il identifie automatiquement les points isolés comme du **bruit** (outliers).

3. Visualisation 2D et interprétation

Comme nos données ont souvent plus de 2 dimensions (ex: température, pression, humidité, temps), on utilise souvent une technique de réduction de dimension comme l'**ACP (Analyse en Composantes Principales)** pour projeter les clusters sur un graphique 2D.

Interprétation :

- **Silhouette Score :** Mesure si les clusters sont bien séparés (proche de 1 = excellent).
- **Analyse des centroïdes :** On regarde les caractéristiques moyennes de chaque groupe pour comprendre ce qu'ils représentent physiquement.

4. Exercices pratiques

Exercice 1 : Utiliser un algorithme de clustering (Python)

Voici comment appliquer K-means avec `Scikit-learn` :

Python
```
from sklearn.cluster import KMeans
import pandas as pd

# 1. Charger ou créer des données (sans étiquettes y)
X = [[1, 2], [1, 4], [1, 0], [10, 2], [10, 4], [10, 0]]

# 2. Initialiser K-means avec 2 clusters
kmeans = KMeans(n_clusters=2, random_state=0)

# 3. Entraînement (Fit)
kmeans.fit(X)

# 4. Voir les résultats (à quel groupe appartient chaque point)
print(f"Groupes assignés : {kmeans.labels_}")
print(f"Centres des clusters : \n{kmeans.cluster_centers_}")
```

Exercice 2 : Visualiser les clusters

Pour "voir" les groupes, on utilise des couleurs différentes pour chaque étiquette prédite par l'algorithme.

Python

```
import matplotlib.pyplot as plt

# On récupère les labels
labels = kmeans.labels_

# Visualisation
plt.scatter([p[0] for p in X], [p[1] for p in X], c=labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red', marker='X', label='Centroïdes')
plt.title("Visualisation des Clusters K-means")
plt.legend()
plt.show()
```

Le clustering est souvent utilisé **avant** la régression. Par exemple, si vous avez des données expérimentales sur un transfert de chaleur, vous pouvez d'abord faire un clustering pour séparer les données "Bruitées" des données "Utiles", puis entraîner votre ANN ou votre SVR uniquement sur le cluster de données stables.

La **Méthode du Coude** (*Elbow Method*) est la technique standard pour répondre à la question : « Combien de groupes (K) se cachent réellement dans mes données ? »

C'est une étape cruciale car, avec K-means, si vous choisissez un K au hasard, vos résultats n'auront aucune valeur scientifique.

1. Le concept : L'Inertie

Pour comprendre cette méthode, il faut définir l'Inertie (ou WCSS - Within-Cluster Sum of Squares).

L'inertie mesure la somme des distances au carré entre chaque point et le centre de son cluster.

- **Si l'inertie est élevée :** Les points sont loin de leurs centres (les clusters sont mal définis).
- **Si l'inertie est faible :** Les points sont très serrés autour de leurs centres (les clusters sont denses).

2. Comment lire le graphique du "Coude" ?

L'idée est de tracer l'évolution de l'inertie en fonction du nombre de clusters K.

1. Quand K augmente, l'inertie diminue forcément (plus il y a de centres, plus les points sont proches d'un centre).
2. Au début, l'inertie chute brutalement.
3. À partir d'un certain point, l'ajout d'un nouveau cluster ne fait presque plus baisser l'inertie.
4. **Le point de rupture**, là où la courbe change de pente (forme un angle), est appelé le **"Coude"**. C'est le nombre optimal de clusters.

3. Mise en œuvre avec Python

Voici le code pour générer ce graphique et choisir votre K :

Python

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Liste pour stocker les valeurs d'inertie
inerties = []
K_range = range(1, 11) # On teste de 1 à 10 clusters

for k in K_range:
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(X) # X est votre dataset
    inerties.append(model.inertia_)

# Tracé du graphique
plt.figure(figsize=(8, 5))
plt.plot(K_range, inerties, 'bx-')
plt.xlabel('Nombre de clusters (K)')
plt.ylabel('Inertie (WCSS)')
plt.title('Méthode du Coude pour déterminer K optimal')
plt.grid(True)
plt.show()
```