**ПРАКТИЧНА РОБОТА №4**

**Кластеризація**

Мета: Закріпити основні поняття використання моделей кластеризації в машинному навчанні.

Завдання:

1. Виконати кластеризацію для набору даних iris методом k-mean++ для визначення сорту ірисів.

**Хід виконання.**

Крок 1.Встановлення бібліотек

Спершу потрібно встановити необхідні бібліотеки, якщо їх немає:

pip install numpy pandas scikit-learn matplotlib

Крок 2.Імпорт необхідних бібліотек

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

Крок 3: Створення набору даних.

Можна використовувати реальний набір даних для кластеризації методом **k-means++**. Один із популярних реальних наборів даних, доступний у бібліотеці scikit-learn, — це **Iris** dataset, який містить дані про різні види квітів Iris.



Результат:



Набір даних **Iris** — це один із найвідоміших і найбільш використовуваних наборів даних для навчання і тестування алгоритмів машинного навчання. Він містить інформацію про різні види квітів **Iris** і використовується для класифікації видів за їхніми фізичними ознаками.

**Опис набору даних Iris:**

* **Кількість зразків**: 150
* **Кількість ознак**: 4
* **Кількість класів (видів)**: 3 (Iris-setosa, Iris-versicolor, Iris-virginica)
* **Тип завдання**: Класифікація (або кластеризація для навчання без учителя)

**Ознаки (фічі):**

1. **Довжина чашолистка** (*sepal length*) — вимірюється в сантиметрах.
2. **Ширина чашолистка** (*sepal width*) — вимірюється в сантиметрах.
3. **Довжина пелюстки** (*petal length*) — вимірюється в сантиметрах.
4. **Ширина пелюстки** (*petal width*) — вимірюється в сантиметрах.

**Класи (мітки):**

Набір даних містить три різні види квітки Iris, представлені як цільові змінні (класи):

1. **Iris-setosa**
2. **Iris-versicolor**
3. **Iris-virginica**

**Призначення:**

Набір даних Iris зазвичай використовується для тестування алгоритмів класифікації, кластеризації та візуалізації. Він є популярним завдяки простоті та малому розміру, що робить його ідеальним для навчання початківців у галузі машинного навчання.

**Структура даних:**

Набір складається із 150 рядків (квіток), де кожен рядок містить значення 4 ознак (довжина і ширина чашолистка та пелюстки) та мітку класу, що вказує на вид квітки.

**Приклад:**



Крок 4. Створюємо модель машинного навчання:

*# Може виникнути помилка або попередження через особливості використання бібліотеки* ***scikit-#learn*** *разом із MKL (Math Kernel Library) на Windows. Це типова проблема для багатопотокових #обчислень, яка стосується визначення кількості фізичних ядер процесора та можливої витоку #пам'яті при використанні алгоритму* ***KMeans****.*

*#Щоб уникнути помилки встанлвлюємо кількість потоків -1*

import os

os.environ["OMP\_NUM\_THREADS"] = "1"

*#Якщо виникає попередження можна його ігнорувати*

import warnings

warnings.filterwarnings("ignore", category=UserWarning)

#Визначаємо модель

kmeans = KMeans(n\_clusters=3, init='k-means++', n\_init=10, max\_iter=300, random\_state=42)

*# Навчаємо модель*

kmeans.fit(X\_scaled)

*# Отримуємо передбачення (до якого кластера належить кожна квітка)*

clusters = kmeans.labels\_

 Крок 5. Візуалізуємо результат:

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=clusters, cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 2], s=300, c='red', label='Centroids')

plt.title('K-means++ Clustering on Iris Dataset')

plt.xlabel('Feature 1')

plt.ylabel('Feature 3')

plt.legend()

plt.show()

Команда plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red', label='Centroids') використовується для відображення центроїдів кластерів на графіку, побудованому за допомогою **Matplotlib**. Ось детальний розбір цієї команди:

**Розбір команди**

1. **plt.scatter()**:
	* Це функція для побудови розсіювання (scatter plot) у Matplotlib. Вона використовується для візуалізації розподілу даних у двовимірному просторі.
2. **kmeans.cluster\_centers\_**:
	* Це атрибут об'єкта KMeans, який містить координати центроїдів кластерів. Кожен центроїд — це середнє значення всіх точок у відповідному кластері.
3. **[:, 0] і [:, 1]**:
	* [:, 0] вибирає перший стовпець з масиву kmeans.cluster\_centers\_, що відповідає координаті по осі X (першій ознаці, наприклад, **Sepal Length**).
	* [:, 1] вибирає другий стовпець, що відповідає координаті по осі Y (другій ознаці, наприклад, **Sepal Width**).
	* Це означає, що ми беремо координати центроїдів у двовимірному просторі.
4. **s=300**:
	* Цей параметр визначає розмір маркерів центроїдів на графіку. Значення 300 означає, що маркери будуть великими, що допоможе їх легше помітити.
5. **c='red'**:
	* Цей параметр задає колір маркерів. У даному випадку центроїди будуть відображатися червоним кольором.
6. **label='Centroids'**:
	* Це текст, який буде показаний у легенді графіка для маркерів центроїдів. Легенда дозволяє зрозуміти, що ці точки представляють центри кластерів.



Крок 5. Використаємо цю модель для віднесення нової квітки до потрібного кластеру (визначимо вид ірису за його ознаками)

* Підготуйте нову квітку у форматі, який відповідає формату ваших навчальних даних.
* Використовуйте метод predict для передбачення кластера, до якого належить нова квітка.

# Підготовка нової квітки (наприклад, 5.1 см, 3.5 см, 1.4 см, 0.2 см)

new\_flower = [[5.1, 3.5, 1.4, 0.2]]

# Стандартизуємо нову квітку

new\_flower\_scaled = scaler.transform(new\_flower)

# Передбачення кластера для нової квітки

predicted\_cluster = kmeans.predict(new\_flower\_scaled)

print("Передбачений кластер для нової квітки:", predicted\_cluster[0])



Відобразимо нову квітку на графіку:

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=clusters, cmap='viridis')

plt.scatter(new\_flower\_scaled[0, 0], new\_flower\_scaled[0, 1], s=200, c='blue', label='New Flower', edgecolor='black')

plt.title('K-means++ Clustering on Iris Dataset')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()



Крок 6. Оцінити якість прогнозування.

Для цього можна використати метрику silhouette\_score

from sklearn.metrics import silhouette\_score

Вона вимірює, наскільки добре кожен об'єкт у кластері відокремлений від інших кластерів.

**Як працює Silhouette Score**

1. **Розрахунок значення**:
	* Для кожної точки обчислюється два значення:
		+ **a(i)** — середня відстань до всіх інших точок у тому ж кластері. Це вимірює, наскільки близька точка до інших точок свого кластера.
		+ **b(i)** — середня відстань до всіх точок у найближчому кластері (не в своєму). Це вимірює, наскільки далеко точка від найближчого кластера.
	* Silhouette Score для точки iii визначається за формулою: 

В Python:

silhouette\_plus = silhouette\_score(X\_scaled, clusters)

silhouette\_plus

Отримаємо:

0.45994823920518635

**Інтерпретація**

* **Високий Silhouette Score** (близько до 1): Кластери добре розділені та компактні. Це вказує на те, що точки у кожному кластері більш схожі одна на одну, ніж на точки з інших кластерів.
* **Низький Silhouette Score** (близько до 0): Це вказує на те, що кластери перекриваються або що точки близькі до межі між кластерами.
* **Негативний Silhouette Score**: Це свідчить про те, що точки можуть бути неправильно кластеризовані, оскільки вони ближче до інших кластерів.

**Самостійно:**

* 1. **Виконайте кластеризацію для цього набору одним з інших методів. Порівняйте якість кластеризації з методом k-mean++.**

**'random'**: У цьому випадку алгоритм вибирає випадкові точки з ваших даних як початкові центри кластерів. Це може призвести до поганих результатів, якщо точки вибрані невдало.

**Agglomerative Clustering.** Агломеративна кластеризація побудована на ідеї "зверху вниз". Вона починає з кожного об'єкта в окремому кластері і об'єднує їх у більші кластери.

**DBSCAN (Density-Based Spatial Clustering of Applications with Noise).** Цей метод кластеризує дані на основі їхньої щільності.

**Spectral Clustering.** Цей метод використовує власні вектори матриці подібності для виконання кластеризації. Він підходить для кластеризації даних з нелінійними межами.

# Кластеризація за допомогою різних методів

kmeans = KMeans(n\_clusters=3, init='random', n\_init=10, max\_iter=300, random\_state=42)

kmeans.fit(X\_scaled)

agglo = AgglomerativeClustering(n\_clusters=3)

agglo.fit(X\_scaled)

dbscan = DBSCAN(eps=0.5, min\_samples=5)

dbscan.fit(X\_scaled)

spectral = SpectralClustering(n\_clusters=3, affinity='nearest\_neighbors')

spectral.fit(X\_scaled)

# Отримання кластерів

kmeans\_clusters = kmeans.labels\_

agglo\_clusters = agglo.labels\_

dbscan\_clusters = dbscan.labels\_

spectral\_clusters = spectral.labels\_

* 1. **Спробуйте також реалізувати карту Кохонена для набору даних ІRIS**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.preprocessing import MinMaxScaler

from sklearn\_som.som import SOM

# Завантажуємо набір Iris

iris = load\_iris()

data = iris.data # Використовуємо тільки ознаки, без міток класів

labels = iris.target # Мітки класів (0 = setosa, 1 = versicolor, 2 = virginica)

# Масштабуємо дані до [0, 1]

scaler = MinMaxScaler()

data = scaler.fit\_transform(data)

# Створюємо та навчаємо SOM (10x10 нейронів, 2 виміри)

som = SOM(m=10, n=10, dim=4)

som.fit(data, epochs=500)

# Отримуємо переможців (координати на карті SOM)

winners = np.array([np.unravel\_index(i, (10, 10)) for i in som.predict(data)])

# Візуалізація кластерів

plt.figure(figsize=(6, 6))

plt.scatter(winners[:, 1], winners[:, 0], c=labels, cmap='viridis', edgecolors='k', s=100)

plt.colorbar(label="Класи Iris (0 = setosa, 1 = versicolor, 2 = virginica)")

plt.gca().invert\_yaxis() # Інвертуємо вісь Y для коректного відображення

plt.title("Самоорганізуюча карта (SOM) для набору даних Iris")

plt.show()

Поясніть, що відкладено по осях.

**Порівняйте якість кластеризації за допомогою карти Кохонена та інших методів.**