**MINISTERUL EDUCAŢIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Programul de studii: Tehnologia informației**



**RAPORT**

**Disciplina „Inteligenta Artificiala”**

**Tema:** **Utilizarea diferitor algoritmi de clusterizare pe diferite tipuri de date. Vizualizarea datelor**

|  |  |  |
| --- | --- | --- |
| **Student(ă):** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Vlași**ț**chi Ștefan , TI-216** |
|  |  |  |
| **Coordonator universitate:** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Viorel Rusu , asist.univ.** |

**Chișinău, 2024**

**Sarcini practice:**

1. **Algoritmul k-Means (Algoritmi partiționali)**:

Concept: Algoritmul k-Means este un algoritm de clusterizare partițional care împarte setul de date în k grupuri, unde fiecare punct de date aparține celui mai apropiat centroid.

Pași de bază:

1. Inițializarea a k centroizi aleatori în spațiul de caracteristici.
2. Atribuirea fiecărui punct de date celui mai apropiat centroid.
3. Recalcularea centroizilor pe baza punctelor atribuite.
4. Repetarea pașilor 2 și 3 până când nu mai există schimbări semnificative în atribuirea punctelor.

Set de date: Putem utiliza un set de date precum Iris pentru a ilustra algoritmul.

from sklearn.datasets import load\_iris  
from sklearn.cluster import KMeans  
from sklearn.metrics import silhouette\_score  
import matplotlib.pyplot as plt  
  
# Încărcăm setul de date Iris  
iris = load\_iris()  
X = iris.data  
  
# Implementăm algoritmul k-Means  
kmeans = KMeans(n\_clusters=3, random\_state=42)  
kmeans.fit(X)  
  
# Evaluăm rezultatele folosind Silhouette Score  
kmeans\_silhouette\_score = silhouette\_score(X, kmeans.labels\_)  
print("Silhouette Score for K-Means:", kmeans\_silhouette\_score)  
  
# Vizualizarea rezultatelor pentru algoritmul k-Means  
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels\_, cmap='viridis')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.title('Clustering by K-Means')  
plt.show()

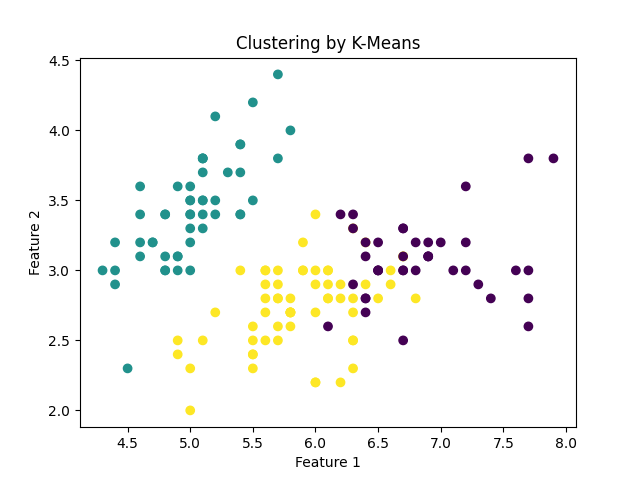




Figura 1 – Clustering by K-Means

1. **Algoritmul Agglomerative Clustering (Algoritmi ierarhici)**:

Concept: Algoritmul aglomerativ începe cu fiecare punct de date ca un cluster separat și îi unește treptat pe cei mai apropiați, formând o ierarhie de clustere.

Pași de bază:

1. Inițializarea fiecărui punct de date ca un cluster individual.
2. Calcularea matricei de distanțe între clustere.
3. Unirea celor două clustere cele mai apropiate.
4. Repetarea pașilor 2 și 3 până când toate punctele sunt unite într-un singur cluster.

Set de date: Putem folosi setul de date Iris sau alt set de date adecvat.

from sklearn.cluster import AgglomerativeClustering  
from sklearn.metrics import silhouette\_score  
import matplotlib.pyplot as plt  
  
# Implementăm algoritmul Agglomerative Clustering  
agglomerative = AgglomerativeClustering(n\_clusters=3)  
agglomerative.fit(X)  
  
# Evaluăm rezultatele folosind Silhouette Score  
agglomerative\_silhouette\_score = silhouette\_score(X, agglomerative.labels\_)  
print("Silhouette Score for Agglomerative Clustering:", agglomerative\_silhouette\_score)  
  
# Vizualizarea rezultatelor pentru algoritmul Agglomerative Clustering  
plt.scatter(X[:, 0], X[:, 1], c=agglomerative.labels\_, cmap='viridis')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.title('Clustering by Agglomerative Clustering')  
plt.show()

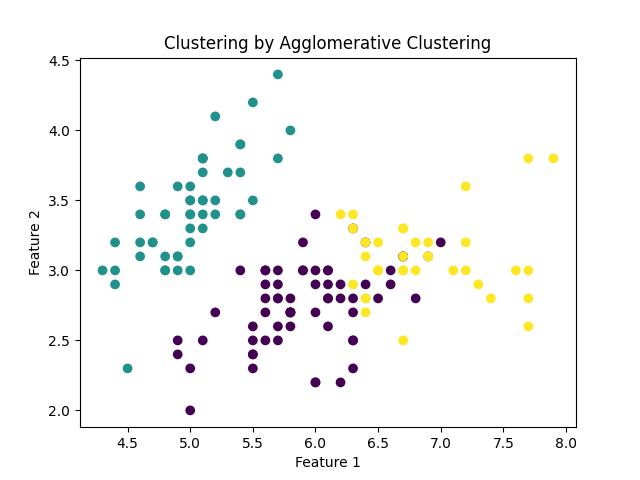


Figura 2 - Clustering by Agglomerative Clustering

1. **Algoritmul DBSCAN (Algoritmi bazate pe densitate)**:

Concept: DBSCAN (Density-Based Spatial Clustering of Applications with Noise) identifică regiunile dense de puncte de date, marcate ca clustere, separate de regiuni mai puțin dense.

Pași de bază:

1. Se alege un punct de date nevizitat și se explorează vecinii săi.
2. Dacă numărul minim de vecini este îndeplinit, se formează un cluster și se extinde pentru a include vecinii săi.
3. Se continuă procesul pentru toți vecinii din cluster.
4. Se repetă pașii 1-3 până când toate punctele sunt atribuite unui cluster sau sunt considerate zgomot.

Set de date: DBSCAN funcționează bine pe seturi de date cu densitate variabilă, cum ar fi seturile de date cu forme complexe sau zgomot.

from sklearn.cluster import DBSCAN  
import matplotlib.pyplot as plt  
  
# Implementăm algoritmul DBSCAN  
dbscan = DBSCAN(eps=0.5, min\_samples=5)  
dbscan.fit(X)  
  
# Evaluăm rezultatele folosind Silhouette Score  
# Notă: DBSCAN nu întoarce etichetele clusterele, astfel că Silhouette Score nu poate fi calculat direct.  
# Vom evalua numărul de clustere și folosirea rezultatelor pentru a face interpretări.  
unique\_labels = set(dbscan.labels\_)  
num\_clusters\_dbscan = len(unique\_labels) - (1 if -1 in dbscan.labels\_ else 0)  
print("Number of clusters found by DBSCAN:", num\_clusters\_dbscan)  
  
# Vizualizarea rezultatelor pentru algoritmul DBSCAN  
plt.scatter(X[:, 0], X[:, 1], c=dbscan.labels\_, cmap='viridis')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.title('Clustering by DBSCAN')  
plt.show()

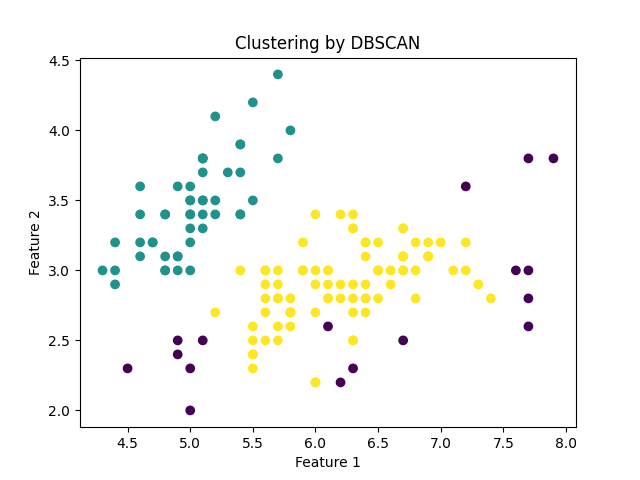




Figura 3 -Clustering by DBSCAN

1. **Algoritmul Gaussian Mixture Model (GMM):**

Concept: Gaussian Mixture Model (GMM) este un algoritm de clustering bazat pe modele probabiliste. În loc să asigneze punctele de date la un singur cluster, GMM atribuie probabilități fiecărui punct de date pentru a aparține fiecărui cluster, având în vedere distribuția gaussiană a datelor.

Pași de bază:

1. Inițializarea modelului GMM cu un număr dat de componente.
2. Estimarea parametrilor modelului, cum ar fi medii, covarianțe și ponderi pentru fiecare componentă.
3. Atribuirea fiecărui punct de date la cea mai probabilă componentă a modelului GMM.
4. Actualizarea parametrilor modelului pe baza atribuirii punctelor de date.
5. Repetarea pașilor 3 și 4 până când modelul converge.

Set de date: Putem utiliza un set de date precum Iris sau alt set de date adecvat pentru a ilustra algoritmul.

from sklearn.mixture import GaussianMixture  
from sklearn.datasets import load\_iris  
  
iris = load\_iris()  
X = iris.data  
  
# Implementăm algoritmul GMM  
gmm = GaussianMixture(n\_components=3, random\_state=42)  
gmm.fit(X)  
  
# Vizualizăm rezultatele clusteringului  
plt.scatter(X[:, 0], X[:, 1], c=gmm.predict(X), cmap='viridis')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.title('Clustering by Gaussian Mixture Model')  
plt.show()  
  
# Afișarea numărului de clustere estimat de modelul GMM  
num\_clusters\_gmm = gmm.n\_components  
print("Number of clusters found by GMM:", num\_clusters\_gmm)

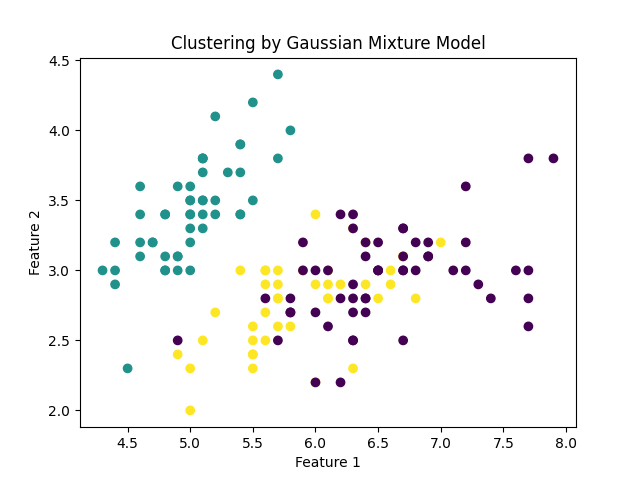




Figura 4 - Clustering by Gaussian Mixture ModelConcluzie

**Concluzie**

Compararea algoritmilor de clusterizare și evaluarea aplicabilității lor în diferite scenarii pot oferi o perspectivă asupra modului în care aceștia pot fi folosiți în diverse domenii și contexte. Iată o concluzie care compară algoritmii k-Means, Agglomerative Clustering, DBSCAN și Gaussian Mixture Model (GMM) și sugerează utilizările lor potențiale:

1. **k-Means:**

- Algoritm de clusterizare partițională care împarte setul de date în k clustere.

- Folosit în mod obișnuit în aplicații care necesită o divizare clară a datelor în clustere distincțe, cum ar fi segmentarea clienților în marketing sau analiza comportamentului utilizatorilor în domeniul web.

1. Agglomerative Clustering;

- Algoritm de clusterizare ierarhic care construiește o ierarhie de clustere.

- Utilizat când există structuri ierarhice în date sau când se dorește o analiză a relațiilor între clustere.

- Potrivit pentru analiza structurilor biologice sau a relațiilor între documente în domeniul prelucrării limbajului natural.

1. DBSCAN:

- Algoritm de clusterizare bazat pe densitate care identifică regiuni dense de puncte de date.

- Ideal pentru seturi de date cu forme complexe sau zgomot, unde densitatea poate varia.

- Utilizat în detecția anomaliilor, identificarea grupurilor de puncte de date apropiate în spațiul de caracteristici, cum ar fi datele geospațiale sau datele de monitorizare a rețelelor.

1. Gaussian Mixture Model (GMM):

- Algoritm de clusterizare bazat pe modele probabiliste care atribuie probabilități fiecărui punct de date pentru a aparține unui cluster.

- Potrivit pentru seturi de date cu distribuții complexe, unde punctele de date pot aparține mai multor clustere.

- Utilizat în recunoașterea de modele, segmentarea imaginilor sau analiza datelor biometrice.

Fiecare algoritm de clusterizare are avantaje și dezavantaje, iar alegerea potrivită depinde de natura datelor, obiectivele analizei și cerințele aplicației. În general, este important să se evalueze performanța algoritmilor în funcție de criteriile specifice ale problemei, precum și de interpretarea și înțelegerea rezultatelor obținute.