42

Name: Dev Parekh

Class: TE9-B-42

Subject: DWM

EXPERIMENT NO. 4

Title: Implementation of Clustering algorithm (K-means/Agglomerative)

Aim: To implement the K-Means clustering algorithm and visualize the clustering process.

Introduction: K-Means is an unsupervised clustering algorithm that partitions a given dataset into K clusters. The algorithm works iteratively by assigning data points to the nearest cluster mean and updating the cluster centroids until convergence. It is widely used for grouping similar data points and pattern recognition. This algorithm is efficient for large datasets and can be applied in various fields such as image segmentation, anomaly detection, and market segmentation. However, it requires the number of clusters (K) to be predefined and may not work well with non-spherical cluster distributions.

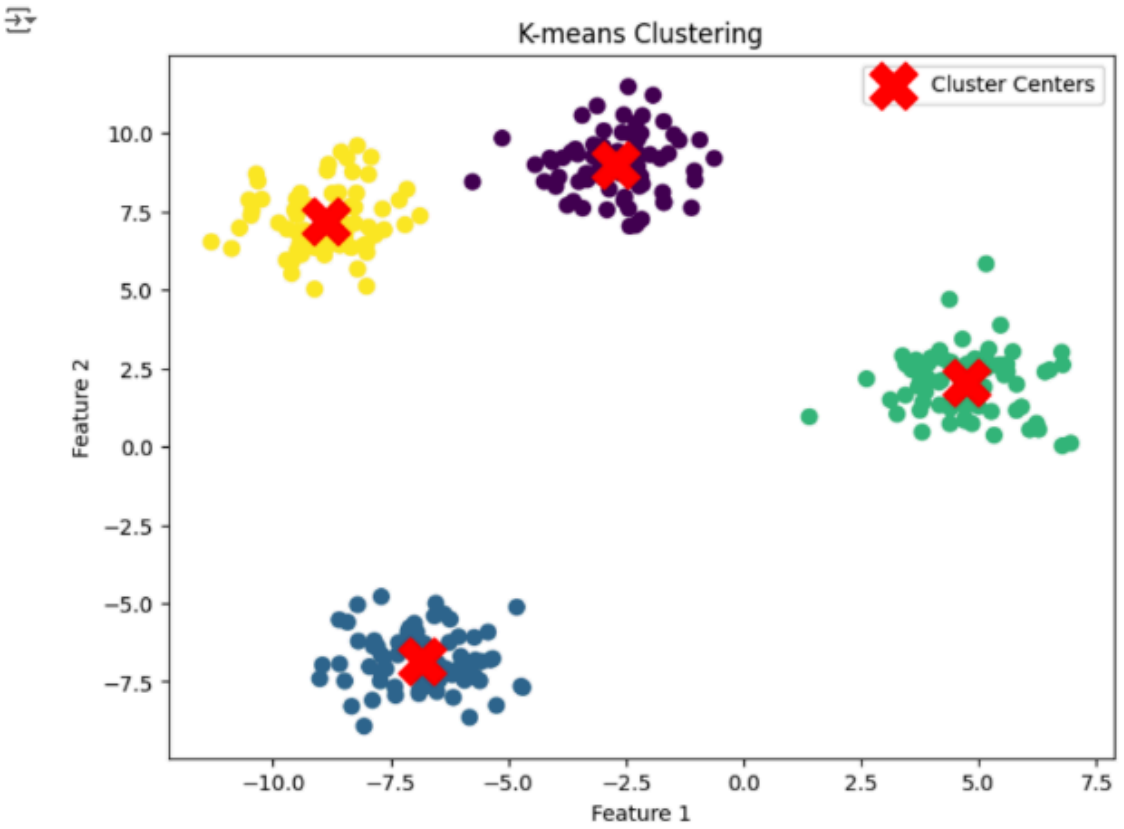
Procedure:

1. **User Input:**
   - Accepts a list of numbers from the user.
   - Takes the number of clusters (*k*) as input.
2. **Initialization:**
   - Randomly selects *k* initial means from the dataset.
3. **Cluster Assignment:**
   - Calculates the distance of each data point from the cluster means.
   - Assigns each point to the nearest cluster.
4. **Mean Update:**
   - Computes new cluster means by averaging the points in each cluster.
5. **Convergence Check:**
   - If the means do not change, the clustering process stops.
   - Otherwise, the process repeats until convergence.
6. **Final Clusters:**
   - Displays the final clusters and their means.
   - Plots a visualization of the clustered data.

Program Code:

```
 import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
# Step 1: Create or load a dataset (using make_blobs for illustration)
# Generating synthetic data with 4 clusters
X, y = make_blobs(n_samples=300, centers=4, random_state=42)
# Step 2: Apply K-means clustering
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)
# Step 3: Get cluster centers and labels
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_
# Step 4: Visualize the clusters
plt.figure(figsize=(8, 6))
# Plot data points with different colors for each cluster
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
# Plot the cluster centers
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', s=500, marker='X', label='Cluster Centers')
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Implementation/Output snap shot:



Conclusion: The K-Means clustering algorithm successfully partitions the data into k clusters based on the distance to the centroids. The final clusters and means are displayed, and a graphical representation of the clustering process is generated. This method is efficient for discovering underlying patterns in data and helps in identifying hidden structures within datasets, making it valuable for data analysis and pattern recognition.

Review Questions:

1. **What is the K-Means clustering algorithm, and how does it work?**

Ans. K-Means is an unsupervised machine learning algorithm used for clustering data into K groups. It works iteratively by:
Randomly selecting K initial centroids.

- Assigning each data point to the nearest centroid.
- Computing new centroids as the mean of all points in each cluster.
- Repeating the process until centroids stabilize.

2. **How do you determine the optimal number of clusters in**

K-Means? Ans. The optimal number of clusters (K) can be determined

using:

- **Elbow Method:** Plots the sum of squared errors (SSE) for different K values; the optimal K is at the "elbow" point.
- **Silhouette Score:** Measures how similar a point is to its assigned cluster versus other clusters.
- **Gap Statistics:** Compares clustering results with random uniform distributions.

3. **What are the common distance metrics used in Agglomerative**

Clustering? Ans. Common distance metrics used in Agglomerative Clustering

include:

- **Euclidean Distance:** Measures the straight-line distance between points.
- **Manhattan Distance:** Measures distance along grid-like paths.
- **Cosine Similarity:** Measures the cosine of the angle between vectors.
- **Mahalanobis Distance:** Considers correlations between variables for clustering.

Github Link: https://github.com/Devp71/DWM