| **CO5** | **Experiment various clustering algorithms to solve real time problems** |
| --- | --- |
| **Task 7a:** | Apply partitioning clustering algorithms for a given dataset. Compare the results and comment on the quality of clustering using evaluation metrics.  **Platform: Google co-lab, Language: Python** |

**Use Case: Iris Flower Species Segmentation for Botanical Research**

**Background:**

A botanical research institute aims to leverage partitioning clustering algorithms to analyze Iris flower species. The researchers have collected data on sepal length, sepal width, petal length, and petal width for a set of Iris flowers. The analysis will be conducted on Google Colab using Python.

**Problem Statement:**

The problem is to analyze the Iris dataset and apply different partitioning clustering algorithms to identify patterns or clusters among the data points representing different iris species. The goal is to compare the results of different partitioning clustering techniques, comment on the quality of clustering using evaluation metrics, and visually inspect the clusters using appropriate graphs.

**Objective**:

1. Apply different partitioning clustering algorithms to group iris data points based on their features.
2. Identify meaningful clusters within the Iris dataset representing different iris species.
3. Compare the results of different partitioning clustering algorithms.
4. Comment on the quality of clustering using evaluation metrics for each algorithm.
5. Visualize the clusters using appropriate graphs.

**Algorithm: Iris Flower Species Segmentation using Partitioning Clustering**

1. Import necessary libraries:

- numpy, pandas, matplotlib.pyplot for data manipulation and visualization

- KMeans, DBSCAN from sklearn.cluster for clustering algorithms

- silhouette\_score, davies\_bouldin\_score from sklearn.metrics for evaluation

- load\_iris from sklearn.datasets for loading the Iris dataset

- StandardScaler from sklearn.preprocessing for data standardization

2. Define functions:

2.1 Function load\_and\_preprocess\_data():

- Load the Iris dataset using load\_iris()

- Standardize the data using StandardScaler

- Return the standardized data and labels

2.2 Function apply\_kmeans(data\_std, n\_clusters):

- Apply K-Means clustering with the specified number of clusters (n\_clusters)

- Return the cluster labels

2.3 Function visualize\_kmeans(data\_std, kmeans\_labels):

- Visualize K-Means clustering results using a scatter plot

- Display the plot

2.4 Function evaluate\_kmeans(data\_std, kmeans\_labels):

- Evaluate K-Means clustering using silhouette score and Davies-Bouldin index

- Print the evaluation metrics

2.5 Function apply\_affinity (data\_std, eps, min\_samples):

- Apply Affinity clustering with specified epsilon (eps) and minimum samples (min\_samples)

- Return the cluster labels

2.6 Function visualize\_ affinity (data\_std, dbscan\_labels):

- Visualize DBSCAN clustering results using a scatter plot

- Display the plot

2.7 Function evaluate\_ affinity (data\_std, dbscan\_labels):

- Evaluate DBSCAN clustering using silhouette score (excluding noise points)

- Print the evaluation metrics

2.8 Function plot\_scatter(data\_std, labels, title):

- Plot a scatter plot for clustering results

- Display the plot with specified title and axis labels

2.9 Follow Same for Birch Clustering

3. Main algorithm (in main()):

- Call load\_and\_preprocess\_data() to obtain standardized data and labels

- Apply K-Means clustering:

- Call apply\_kmeans() with the standardized data and the desired number of clusters

- Call visualize\_kmeans() to visualize the clustering results

- Call evaluate\_kmeans() to evaluate the clustering performance

- Apply Affinity clustering:

- Call apply\_affinity() with the standardized data and specified epsilon and minimum samples

- Call visualize\_ affinity () to visualize the clustering results

- Call evaluate\_ affinity () to evaluate the clustering performance

- Apply Birch clustering:

- Call apply\_birch() with the standardized data and specified epsilon and minimum samples

- Call visualize\_ birch () to visualize the clustering results

- Call evaluate\_ birch () to evaluate the clustering performance

4. Execute the main algorithm:

- Call the main() function to perform the clustering analysis on the Iris dataset.

**Program:**

# Install required libraries

!pip install -q pandas numpy matplotlib scikit-learn

# Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans, AffinityPropagation, Birch

from sklearn.metrics import silhouette\_score, davies\_bouldin\_score, calinski\_harabasz\_score

from sklearn.datasets import load\_iris

# Load Iris dataset

iris = load\_iris()

data = pd.DataFrame(data= np.c\_[iris['data'], iris['target']], columns= iris['feature\_names'] + ['target'])

# Select relevant features for clustering

selected\_features = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

X = data[selected\_features]

# K-means clustering function

def kmeans\_clustering(X, n\_clusters=3):

model = KMeans(n\_clusters=n\_clusters, random\_state=42)

labels = model.fit\_predict(X)

return labels

# Affinity Propagation clustering function

def affinity\_propagation\_clustering(X):

model = AffinityPropagation()

labels = model.fit\_predict(X)

return labels

# Birch clustering function

def birch\_clustering(X, n\_clusters=3):

model = Birch(n\_clusters=n\_clusters)

labels = model.fit\_predict(X)

return labels

# Function to evaluate clustering metrics

def evaluate\_clustering(X, labels, algorithm):

silhouette = silhouette\_score(X, labels)

db\_index = davies\_bouldin\_score(X, labels)

ch\_index = calinski\_harabasz\_score(X, labels)

print(f'Evaluation Metrics for {algorithm}:')

print(f'Silhouette Score: {silhouette}')

print(f'Davies-Bouldin Index: {db\_index}')

print(f'Calinski-Harabasz Index: {ch\_index}\n')

# Function to plot clusters

def plot\_clusters(X, labels, algorithm):

plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=labels, cmap='viridis', marker='o', edgecolors='k')

plt.title(f'{algorithm} Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

# Apply K-means clustering

kmeans\_labels = kmeans\_clustering(X)

evaluate\_clustering(X, kmeans\_labels, 'K-Means Clustering')

plot\_clusters(X, kmeans\_labels, 'K-Means')

# Apply Affinity Propagation clustering

affinity\_labels = affinity\_propagation\_clustering(X)

evaluate\_clustering(X, affinity\_labels, 'Affinity Propagation')

plot\_clusters(X, affinity\_labels, 'Affinity Propagation')

# Apply Birch clustering

birch\_labels = birch\_clustering(X)

evaluate\_clustering(X, birch\_labels, 'Birch Clustering')

plot\_clusters(X, birch\_labels, 'Birch')

Evaluation Metrics for K-Means Clustering:

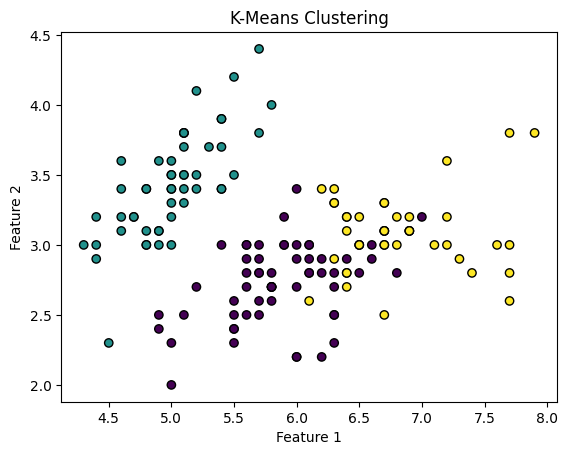
Silhouette Score: 0.5528190123564095

Davies-Bouldin Index: 0.6619715465007465

Calinski-Harabasz Index: 561.62775662962

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/\_kmeans.py:870: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning

warnings.warn(

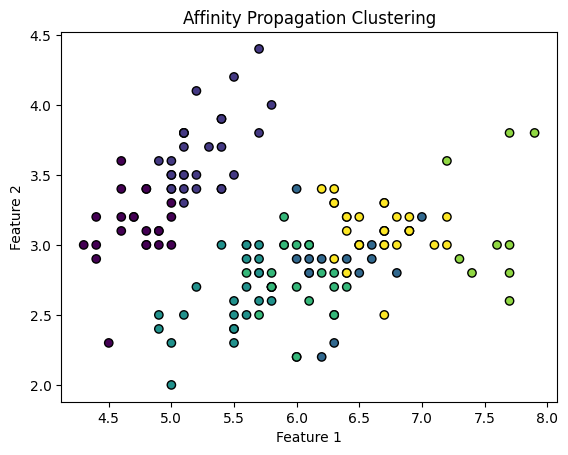


Evaluation Metrics for Affinity Propagation:

Silhouette Score: 0.3474081937055608

Davies-Bouldin Index: 0.9853972233056473

Calinski-Harabasz Index: 443.79711286686637



Evaluation Metrics for Birch Clustering:

Silhouette Score: 0.5019524848046077

Davies-Bouldin Index: 0.625830592433168

Calinski-Harabasz Index: 458.47251055625765

