National University of Computer and Emerging Sciences



Lab Manual

Artificial Intelligence Lab

Department of Computer Science

FAST-NU, Lahore, Pakistan

**K-means, K-medians, K-medoids Clustering Algorithms**

**Objective:**

The objective of this lab manual is to help you understand and implement the K-means, K-medians and K-medoids clustering algorithm using Python.

**Introduction:**

**K-Medians:**

K-median clustering is a method for partitioning a dataset into 𝑘 clusters, where **each cluster is represented by the median of its data points.** The algorithm starts by dividing the dataset into 𝑘 initial groups. Then, it calculates the median for each cluster and assigns each data point to the cluster whose median is closest. This process is repeated iteratively until there is no change in the assignment of data points to clusters.

**K-Means:**

K-means clustering is another technique for dividing a dataset into 𝑘 clusters, but in this case, **each cluster is represented by the mean of its data points.** Like k-median, the algorithm begins by splitting the dataset into 𝑘 initial groups. It then computes the mean for each cluster and assigns each data point to the cluster whose mean is nearest. This process is repeated iteratively until there is no further change in the assignment of data points to clusters.

**K-Mediods:**

K-medoid clustering is a method used to divide a dataset into 𝑘 clusters, where each cluster is represented by its **most centrally located data point, known as the "medoid."** In this technique, the algorithm initially divides the dataset into 𝑘 groups. Then, it calculates the medoid for each group, which is the data point within the group that has the minimum average dissimilarity to all other points in the same group. After computing the medoids, each data point is assigned to the cluster whose medoid is closest. This process is repeated iteratively until no further changes occur in the assignment of data points to clusters.

**Lab Outline:**

1. Import the required libraries:

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.datasets import make\_blobs  from sklearn.preprocessing import StandardScaler |

1. Load and preprocess the dataset

|  |
| --- |
| # Generate a synthetic dataset with 300 samples and 2 features  data , labels= make\_blobs(n\_samples=300, centers=4, n\_features=2, random\_state=42)  # Standardize the dataset  scaler = StandardScaler()  data\_scaled = scaler.fit\_transform(data) |

1. Implement the K-means algorithm

|  |
| --- |
| def initialize\_centroids(data, k):  """Randomly initialize the centroids from the data points."""  # Your implementation here  def compute\_distances(data, centroids):  """Compute the distances between each data point and centroids."""  # Your implementation here  def assign\_clusters(distances):  """Assign each data point to the closest centroid."""  # Your implementation here  def update\_centroids(data, clusters, k):  """Update the centroids by computing the mean of the points in each cluster."""  # Your implementation here  def k\_means(data, k, max\_iterations=100):  """Implement the K-means clustering algorithm."""  # Your implementation here |

1. Evaluate the results

|  |
| --- |
| # Choose the number of clusters, K  k = 4  # Run the K-means algorithm  centroids, clusters = k\_means(data\_scaled, k)  # Compute the total within-cluster sum of squares  wcss = np.sum([np.sum(np.square(data\_scaled[clusters == i] - centroids[i])) for i in range(k)])  print("Total within-cluster sum of squares: ", wcss) |

1. Visualize the clusters

|  |
| --- |
| # Plot the dataset with the assigned clusters and centroids  plt.scatter(data\_scaled[:, 0], data\_scaled[:, 1], c=clusters, cmap='viridis')  plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='x')  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  plt.title('K-means Clustering Results')  plt.show() |

1. For K-Medians:

|  |
| --- |
| def initialize\_centroids(data, k):  """Initialization - Split the data into k initial clusters"""  # Your implementation here  def compute\_distances(data, centroids):  """Compute the distance of each data point from the medians of the clusters. """  # Your implementation here  def assign\_clusters(distances):  """Assign each data point to the cluster whose median is closest. """  # Your implementation here  def update\_centroids(data, clusters, k):  """Compute the median of each cluster based on the assigned data points. """  # Your implementation here  def k\_median(data, k, max\_iterations=100):  """  Implementing the k-median clustering algorithm.  """  # Your implementation here |

1. For K-Medoids

|  |
| --- |
| def initialize\_medoids(data, k):  """Randomly initialize the medoids from the data points."""  # Your implementation here  def compute\_dissimilarities(data, medoids):  """Compute the dissimilarities between each data point and medoids."""  # Your implementation here  def assign\_clusters(dissimilarities):  """Assign each data point to the closest medoid."""  # Your implementation here  def update\_medoids(data, clusters, k):  """Update the medoids by selecting the data point with the minimum sum of dissimilarities in each cluster."""  # Your implementation here  def k\_medoids(data, k, max\_iterations=100):  """Implement the K-medoids clustering algorithm."""  # Your implementation here |

**Make sure to complete all K mean and Kmedians and K medoids with their visualization and submit a single notebook**.