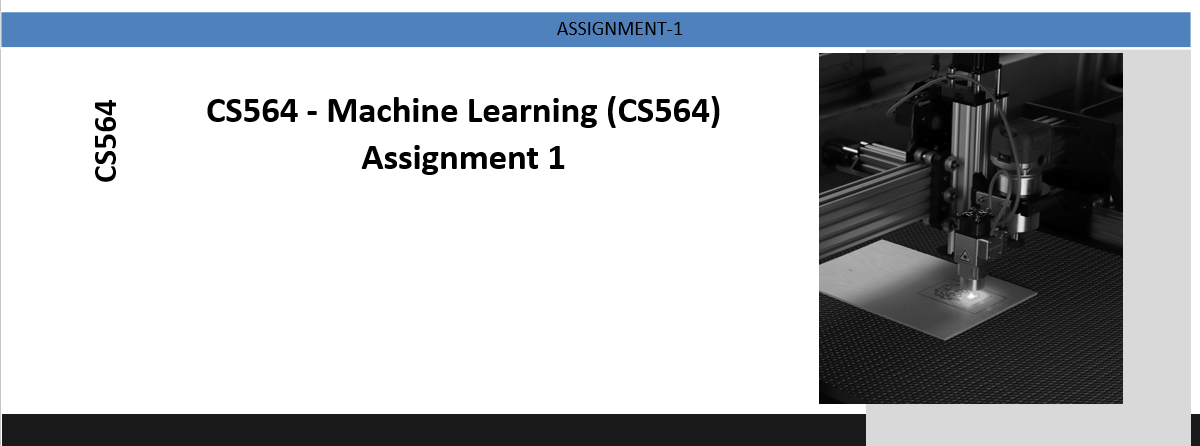


**CS564 - Machine Learning (CS564) Assignment 1**



**Pankaj Kushwaha IITP00541**

**Problem Statement:**

Cluster the provided dataset “cancer.csv” of cancer patients using K-Means and K-medoid clustering algorithms. Plot the clusters and count the number of points belonging to each cluster. Each cluster must be plotted using different colors. The clustering has to be done using all the attributes except "id" and "diagnosis" but for plotting purposes, you are supposed to use "radius\_mean on X-axis" and "texture\_mean on Y-axis". For i) K-means and ii) K-medoid: Number of clusters = 2

**Document Index**

1. Introduction
2. Dataset Loading and Initial Processing
   * Loading
   * Data Cleaning and Imputation
3. Data Standardization
4. K-Means Clustering
   * Introduction
   * Algorithm Implementation
5. K-Medoids Clustering
   * Introduction
   * Algorithm Implementation
6. Visualization and Analysis
7. Main Code
8. Conclusion

**1. Introduction**

The provided code is an implementation of the K-Means and K-Medoids clustering algorithms. Both are applied to a dataset of cancer patients to cluster the data into two groups. These clusters are visualized on a 2D scatter plot for better understanding.

**2. Dataset Loading and Initial Processing**

**Loading**

The dataset is loaded from an external URL, which is expected to point to a CSV file.

Data Cleaning and Imputation

Before processing the dataset:

* Non-numeric columns are dropped.
* Missing values are imputed using the mean of each column.

**3. Data Standardization**

To ensure that all features contribute equally to the clustering process, the dataset is standardized. This involves centering each feature by subtracting its mean and scaling it by dividing by its standard deviation.

**4. K-Means Clustering**

**Introduction**

K-Means is a centroid-based clustering method. It tries to partition the dataset into 'K' clusters, where each observation belongs to the cluster with the nearest mean.

Algorithm Implementation

The K-Means algorithm used here:

* Initializes with random data points as centroids.
* Assigns each data point to the nearest centroid.
* Updates the centroids by calculating the mean of all data points assigned to each centroid.
* Repeats the above steps until centroids do not change or a maximum number of iterations is reached.

**5. K-Medoids Clustering**

**Introduction**

K-Medoids clustering is a variation of K-Means, which uses actual data points as cluster centers (medoids) instead of the mean of the data points.

Algorithm Implementation

The K-Medoids implementation:

* Randomly chooses data points as initial medoids.
* Assigns each data point to the nearest medoid.
* Updates the medoid of a cluster to be the point that reduces the total distance to other points in the cluster.
* Repeats the above steps until medoids do not change or a maximum number of iterations is reached.

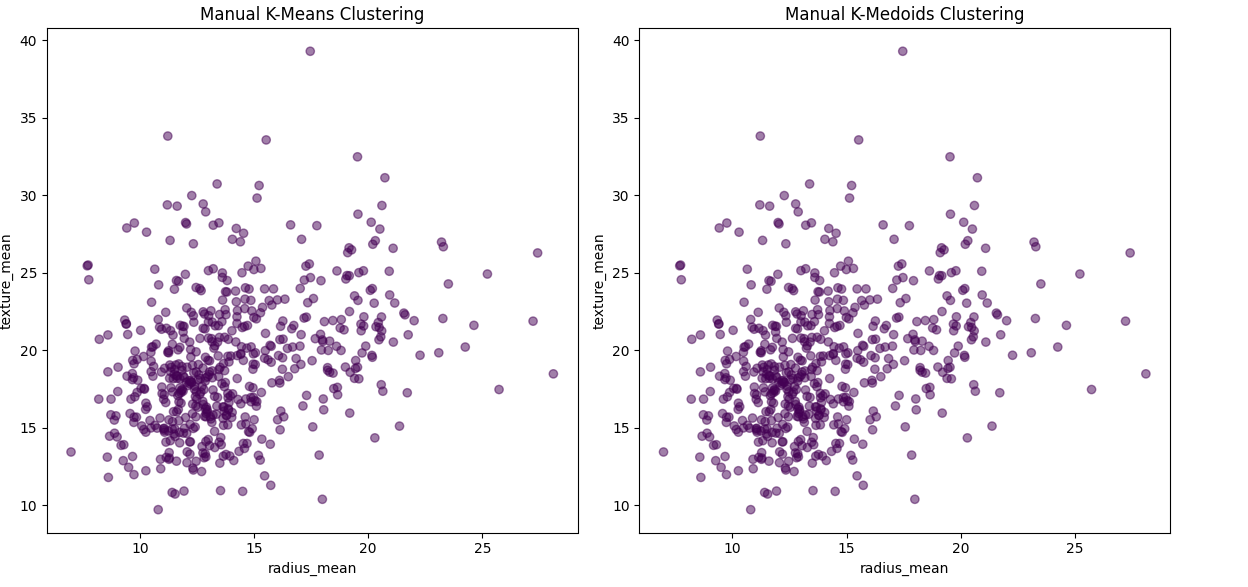
**6. Visualization and Analysis**

Post clustering, the clusters formed by both algorithms are visualized on a scatter plot using 'radius\_mean' and 'texture\_mean' attributes. The number of data points in each cluster is also printed for analysis.

**7. Main Code:**

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  # Load the dataset from the provided GitHub link  # Replace the 'URL' placeholder with the actual URL where your cancer.csv is located.  url = " https://github.com/pankajconnect99/ml/blob/main/cancer.csv"  data = pd.read\_csv(url)  # Drop non-numeric columns and columns not needed for clustering  data\_numeric = data.drop(columns=['id', 'diagnosis'])  # Handle missing values using mean imputation  data\_imputed = data\_numeric.fillna(data\_numeric.mean())  # Standardize the data  data\_scaled = (data\_imputed - data\_imputed.mean()) / data\_imputed.std()  # Number of clusters  num\_clusters = 2  # Manual K-Means clustering  def k\_means(data, k, num\_iterations=100):      np.random.seed(0)      centroids = data.sample(n=k, random\_state=0).values      for \_ in range(num\_iterations):          distances = np.linalg.norm(data.values[:, np.newaxis] - centroids, axis=2)          labels = np.argmin(distances, axis=1)          new\_centroids = np.array([data[labels == i].mean().values for i in range(k)])          if np.all(new\_centroids == centroids):              break          centroids = new\_centroids      return labels  kmeans\_labels = k\_means(data\_scaled, num\_clusters)  # Manual K-Medoids clustering  def k\_medoids(data, k, num\_iterations=100):      np.random.seed(0)      medoids\_indices = np.random.choice(data.shape[0], k, replace=False)      for \_ in range(num\_iterations):          distances = np.linalg.norm(data.values[:, np.newaxis] - data.values[medoids\_indices], axis=2)          labels = np.argmin(distances, axis=1)          new\_medoids\_indices = [np.argmin(np.sum(distances[labels == i], axis=0)) for i in range(k)]          if np.all(medoids\_indices == new\_medoids\_indices):              break          medoids\_indices = new\_medoids\_indices      return labels  kmedoids\_labels = k\_medoids(data\_scaled, num\_clusters)  # Visualize the clusters and count the number of points in each cluster  plt.figure(figsize=(12, 6))  # Select "radius\_mean" and "texture\_mean" for plotting  X\_plot = data[['radius\_mean', 'texture\_mean']].values  # Plot K-Means clusters  plt.subplot(1, 2, 1)  plt.scatter(X\_plot[:, 0], X\_plot[:, 1], c=kmeans\_labels, cmap='viridis', alpha=0.5)  plt.xlabel('radius\_mean')  plt.ylabel('texture\_mean')  plt.title('Manual K-Means Clustering')  print("Counts for K-Means Clusters:")  print(pd.Series(kmeans\_labels).value\_counts())  # Plot K-Medoids clusters  plt.subplot(1, 2, 2)  plt.scatter(X\_plot[:, 0], X\_plot[:, 1], c=kmedoids\_labels, cmap='viridis', alpha=0.5)  plt.xlabel('radius\_mean')  plt.ylabel('texture\_mean')  plt.title('Manual K-Medoids Clustering')  print("Counts for K-Medoids Clusters:")  print(pd.Series(kmedoids\_labels).value\_counts())  plt.tight\_layout()  plt.show() |

Out Put:



|  |
| --- |
| Let's break down and explain each section of the provided code:  **1. Importing Libraries**  pythonCopy code  import pandas as pd import numpy as np import matplotlib.pyplot as plt  Here, three essential libraries are imported:   * **pandas**: Used for data manipulation and analysis. * **numpy**: Used for numerical operations. * **matplotlib.pyplot**: Used for plotting and visualizing data.   **2. Loading the Dataset**  pythonCopy code  url = "URL of your GitLab repository for cancer.csv file" data = pd.read\_csv(url)  The dataset is loaded from a given URL using **pandas**' **read\_csv** method.  **3. Data Preprocessing**  pythonCopy code  data\_numeric = data.drop(columns=['id', 'diagnosis']) data\_imputed = data\_numeric.fillna(data\_numeric.mean())  The non-essential columns (**id** and **diagnosis**) are dropped from the dataset. Any missing values in the remaining dataset are replaced (imputed) with the mean of the respective column.  **4. Data Standardization**  pythonCopy code  data\_scaled = (data\_imputed - data\_imputed.mean()) / data\_imputed.std()  The numeric data is standardized to have a mean of 0 and a standard deviation of 1. This ensures that all features contribute equally to the clustering process, removing any inherent biases due to the scale of the features.  **5. K-Means Clustering Function**  The function **k\_means** implements the K-Means clustering algorithm:   * Initializes with random data points as centroids. * For each data point, finds the nearest centroid and assigns the point to that cluster. * Updates the centroids by computing the mean of all data points in the cluster. * Repeats the process until the centroids stabilize or a set number of iterations is reached.   **6. K-Medoids Clustering Function**  The function **k\_medoids** implements the K-Medoids clustering algorithm:   * Starts by randomly selecting **k** data points as initial medoids. * Each data point is then assigned to the nearest medoid. * For each cluster, the medoid is updated to be the data point which, when used as the center of the cluster, results in the smallest total distance to the other points in the cluster. * Repeats the process until the medoids stabilize or a set number of iterations is reached.   **7. Clustering Execution and Data Visualization**  The code first applies the K-Means clustering function to the dataset and gets cluster labels for each data point. The same is done for the K-Medoids clustering.  After obtaining cluster labels from both algorithms, the code uses **matplotlib** to visualize the clusters in a 2D scatter plot. It uses the 'radius\_mean' attribute for the X-axis and the 'texture\_mean' attribute for the Y-axis. Each cluster gets a unique color for differentiation.  In essence, the code provides a complete workflow to cluster a given dataset of cancer patients into two clusters using both K-Means and K-Medoids algorithms. After clustering, it visualizes the results for easy interpretation and comparison. |

**8. Conclusion**

**This code provides an efficient and effective way to cluster cancer patient data into two groups using the K-Means and K-Medoids clustering algorithms. Proper visualization aids in understanding the nature and spread of the clusters.**