lOMoARcPSD|17241975

**Experiment 7**

**Aim:** Implementation of Clustering algorithm (K-means/K-medoids)

# Theory:

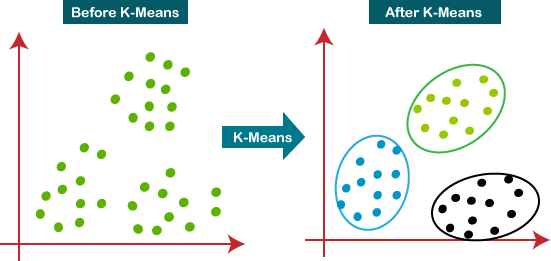
**K-Means Clustering Algorithm:**

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here **K** defines the number of pre-defined clusters that need to be created in the process, as if **K=2**, there will be two clusters, and for **K=3**, there will be three clusters, and so on. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm works as follows:

1. First, we initialize k points, called means, randomly.
2. We categorize each item to its closest mean, and we update the mean’s coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.



# K-Medoids Clustering Algorithm:

K-Medoids (also called as Partitioning Around Medoid) algorithm was proposed in 1987 by Kaufman and Rousseeuw. A medoid can be defined as the point in the cluster, whose dissimilarities with all the other points in the cluster is minimum.

The dissimilarity of the medoid(Ci) and object(Pi) is calculated by using E = |Pi - Ci| The cost in K-Medoids algorithm is given as



# Algorithm:

1. Initialize: select k random points out of the n data points as the medoids.
2. Associate each data point to the closest medoid by using any common distance metric methods.
3. While the cost decreases:

For each medoid m, for each data o point which is not a medoid:

* 1. Swap m and o, associate each data point to the closest medoid, recompute the cost.
  2. If the total cost is more than that in the previous step, undo the swap.

# Program (K-means):

*# Loading the required modules*

*import* matplotlib.pyplot *as* plt

*import* numpy *as* np

*from* scipy.spatial.distance *import* cdist *from* sklearn.datasets *import* load\_digits *from* sklearn.decomposition *import* PCA

*# Defining our function*

*def* kmeans(x, k, no\_of\_iterations):

idx = np.random.choice(len(x), k, replace=*False*)

*# Randomly choosing Centroids*

centroids = x[idx, :] *# Step 1*

*# finding the distance between centroids and all the data points*

distances = cdist(x, centroids, 'euclidean') *# Step 2*

*# Centroid with the minimum Distance*

points = np.array([np.argmin(i) *for* i *in* distances]) *# Step 3*

*# Repeating the above steps for a defined number of iterations # Step 4*

*for* \_ *in* range(no\_of\_iterations): centroids = []

*for* idx *in* range(k):

*# Updating Centroids by taking mean of Cluster it belongs to* temp\_cent = x[points == idx].mean(axis=0) centroids.append(temp\_cent)

centroids = np.vstack(centroids) *# Updated Centroids*

distances = cdist(x, centroids, 'euclidean')

points = np.array([np.argmin(i) *for* i *in* distances])

*return* points

*# Load Data*

data = load\_digits().data pca = PCA(2)

*# Transform the data*

df = pca.fit\_transform(data)

*# Applying our function*

label = kmeans(df, 10, 1000)

*# Visualize the results*

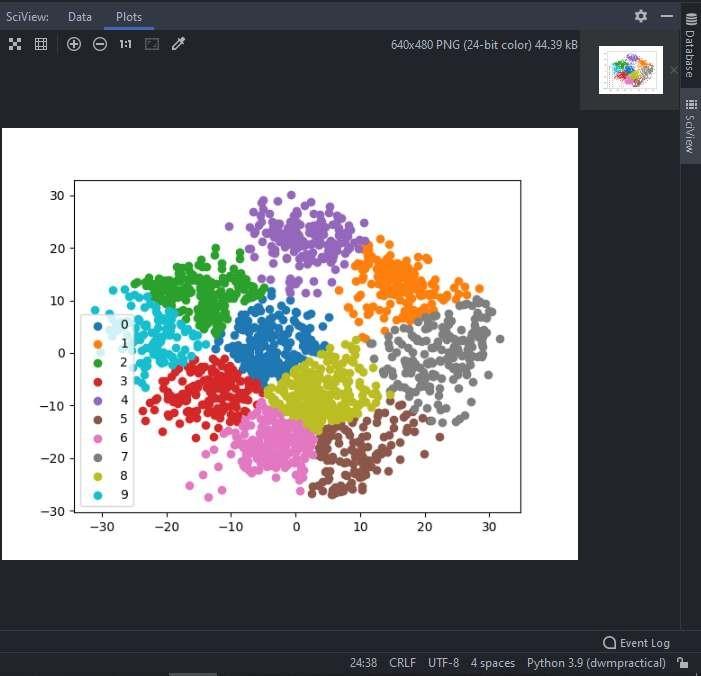
u\_labels = np.unique(label)

*for* i *in* u\_labels:

plt.scatter(df[label == i, 0], df[label == i, 1], label=i) plt.legend()

plt.show()

**Output:**



**Conclusion:** Thus, we successfully implemented clustering algorithm (K-means/K-medoids).