Department of Computer Science and Engineering (Data Science)

Machine Learning – IV

Experiment 8

Name: Umang Kirit Lodaya **SAP ID: 60009200032**

Batch: D11

Aim:

The aim of this lab is to implement and understand the Clustering Using Representatives (CURE) algorithm. CURE is a hierarchical clustering algorithm that is particularly effective for large datasets. It employs a combination of hierarchical clustering and partitioning to efficiently cluster data points.

Theory:

Introduction to CURE Algorithm:

- The CURE algorithm, introduced by Guha, Rastogi, and Shim in 1998, is a partitional clustering algorithm.
- It is designed to handle large datasets efficiently and produces a hierarchical clustering structure.
- The key idea is to represent each cluster by a set of representative points, known as medoids, which are the actual data points rather than centroids.

Algorithm Overview:

- Initialization: Select a random sample of points from the dataset to be used as initial medoids.
- Clustering: Assign each point to the cluster represented by the nearest medoid.
- Medoid Update: Recalculate the medoids to minimize the sum of distances within each cluster.
- Iteration: Repeat the clustering and medoid update steps until convergence.
- Agglomerative Hierarchical Clustering: Combine clusters at each iteration to form a hierarchy.

Step-by-Step Implementation:

- **Step 1: Initialization**
  - Randomly select k data points from the dataset as initial medoids.
  - Store these medoids in a list.
- **Step 2: Assign Points to Clusters**
  - For each data point, assign it to the cluster represented by the nearest medoid.
  - Use a distance metric such as Euclidean distance.
- **Step 3: Update Medoids**
  - For each cluster, calculate the sum of distances between all points and choose the point with the minimum sum as the new medoid.
  - Update the medoid list.
- **Step 4: Convergence Check**
  - Repeat steps 2 and 3 until convergence, i.e., until the medoids do not change significantly.
- **Step 5: Hierarchical Clustering**
  - Apply agglomerative hierarchical clustering to the clusters obtained in the previous steps.
  - Use a linkage criterion like complete linkage.

Implementation Tips:

- Use efficient data structures for distance calculations to speed up the algorithm.
- Experiment with different distance metrics and linkage criteria.
- Monitor convergence by tracking changes in the medoid list.

Lab Experiments to be Performed in This Session:

Execute the CURE algorithm on a dataset to gain insights into its functionality and operation.

NAME: UMANG KIRIT LODAYA

SAP ID: 60009200032

BATCH: D11

CURE ALGORITHM

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/dataset/Comments.txt
/kaggle/input/dataset/data/3clus.txt
/kaggle/input/dataset/data/jain.txt
/kaggle/input/dataset/data/aggregation.txt
```

In [2]:

```
import numpy as np
from scipy.spatial.distance import pdist
import time
from sklearn.metrics.cluster import v_measure_score

import matplotlib.pyplot as plt
```

In [3]:
```
files = {}
with open("/kaggle/input/dataset/Comments.txt", "r") as f:
    for fil in f.readlines():
        file, numRepPoints, alpha, numDesCluster = fil.split()
        files[file] = [int(numRepPoints), float(alpha), int(nu
mDesCluster)]

def plot(data_set, Label_pre):
    plt.title("RESULT")
    scatterColors = ['black', 'blue', 'green', 'yellow', 're
d', 'purple', 'orange', 'brown', 'cyan', 'brown',
                    'chocolate', 'darkgreen', 'darkblue', 'azu
re', 'bisque']
    for i in range(data_set.shape[0]):
        color = scatterColors[Label_pre[i]]
        plt.scatter(data_set[i, 0], data_set[i, 1], marker =
'.', c = color)

    plt.show()
```

In [4]:

```
# RETURNS EUCLIDEAN DISTANCE
def dist(vecA, vecB):
    return np.sqrt(np.power(vecA - vecB, 2).sum())

class CureCluster:
    def __init__(self, id__, center__):
        self.points = center__
        self.repPoints = center__
        self.center = center__
        self.index = [id__]

    def __repr__(self):
        return "SIZE: " + str(len(self.points))

    # COMPUTES AND UPDATES NEW CENTRIOD
    def computeCentroid(self, clust):
        totalPoints_1 = len(self.index)
        totalPoints_2 = len(clust.index)
        self.center = (self.center*totalPoints_1 + clust.cente
r*totalPoints_2) / (totalPoints_1 + totalPoints_2)

    # COMPUTES REPRESENTATIVE POINTS
    def generateRepPoints(self, numRepPoints, alpha):
        tempSet = None
        for i in range(1, numRepPoints+1):
            maxDist = 0
            maxPoint = None
            for p in range(0, len(self.index)):
                if i == 1:
                    minDist = dist(self.points[p,:], self.cent
er)
                else:
                    X = np.vstack([tempSet, self.points[p,
:]])
                    tmpDist = pdist(X)
                    minDist = tmpDist.min()

                if minDist >= maxDist:
                    maxDist = minDist
                    maxPoint = self.points[p,:]

            if tempSet is None:
                tempSet = maxPoint
            else:
                tempSet = np.vstack((tempSet, maxPoint))

        for j in range(len(tempSet)):
            if self.repPoints is None:
                # SHRINKING THE DISTANCE BETWEEN CENTRIOD AND
REP. POINTS
                self.repPoints = tempSet[j,:] + alpha * (self.
center - tempSet[j,:])
            else:
                self.repPoints = np.vstack((self.repPoints, te
mpSet[j,:] + alpha * (self.center - tempSet[j,:])))

    # COMPUTES MIN. DISTANCE BETWEEN REP. POINTS OF 2 CLUSTERS
    def distRep(self, clust):
        distRep = float('inf')
        for repA in self.repPoints:
```

```python
            if type(clust.repPoints[0]) != list:
                repB = clust.repPoints
                distTemp = dist(repA, repB)
                if distTemp < distRep:
                    distRep = distTemp
            else:
                for repB in clust.repPoints:
                    distTemp = dist(repA, repB)
                    if distTemp < distRep:
                        distRep = distTemp
        return distRep

    # MERGES THE 2 CLUSTER, AND RECOMPUTES VALUES
    def mergeWithCluster(self, clust, numRepPoints, alpha):
        self.computeCentroid(clust)
        self.points = np.vstack((self.points, clust.points))
        self.index = np.append(self.index, clust.index)
        self.repPoints = None
        self.generateRepPoints(numRepPoints, alpha)
```

In [5]:

```
# RUNS THE CURE ALGORITHM
def runCURE(data, numRepPoints, alpha, numDesCluster, printClu
sterSize = False):
    # INITIALIZATION
    Clusters = []
    numCluster = len(data)
    numPts = len(data)
    distCluster = np.ones([len(data), len(data)])
    distCluster = distCluster * float('inf')

    # MAKING EACH POINT A CLUSTER
    for idPoint in range(len(data)):
        newClust = CureCluster(idPoint, data[idPoint,:])
        Clusters.append(newClust)

    # CALCULATING DISTANCE BETWEEN EACH POINT (CLUSTER)
    for row in range(0, numPts):
        for col in range(0, row):
            distCluster[row][col] = dist(Clusters[row].center,
Clusters[col].center)

    while numCluster > numDesCluster:
        if np.mod(numCluster, 100) == 0:
            print(f"CURRENT CLUSTER COUNT: {numCluster}")

        # FIND PAIR OF CLOSEST POINTS
        minIndex = np.where(distCluster == np.min(distCluste
r))
        minIndex1 = minIndex[0][0]
        minIndex2 = minIndex[1][0]

        # MERGING BOTH POINTS
        Clusters[minIndex1].mergeWithCluster(Clusters[minIndex
2], numRepPoints, alpha)

        # UPDATING THE DISTANCE BETWEEN CLUSTERS
        for i in range(0, minIndex1):
            distCluster[minIndex1, i] = Clusters[minIndex1].di
stRep(Clusters[i])
        for i in range(minIndex1+1, numCluster):
            distCluster[i, minIndex1] = Clusters[minIndex1].di
stRep(Clusters[i])

        # REMOVE SECOND POINT
        distCluster = np.delete(distCluster, minIndex2, axis=
0)
        distCluster = np.delete(distCluster, minIndex2, axis=
1)
        del Clusters[minIndex2]
        numCluster = numCluster - 1

    print()
    print(f"FINAL CLUSTER COUNT: {numCluster}")
    if printClusterSize:
        print("\nSIZE OF EACH CLUSTER:")
        print()

    # PREDICTING CLUSTER LABEL FOR EACH POINTS
    Label = [0] * numPts
    for i in range(0, len(Clusters)):
```

```python
        if printClusterSize:
            print(f"CLUSTER {i+1} -> {Clusters[i]}")
        for j in range(0, len(Clusters[i].index)):
            Label[Clusters[i].index[j]] = i + 1

    return Label
```

In [6]:
```python
file = "aggregation.txt"
data_set = np.loadtxt(f'/kaggle/input/dataset/data/{file}')
numRepPoints, alpha, numDesCluster = files[file]
print(f"PEFORMING CLUSTERING FOR {file}")
print(f"DESIRED NUMBER OF REP. POINTS: {numRepPoints}")
print(f"DESIRED NUMBER OF CLUSTER: {numDesCluster}")
print(f"SHRINKING FACTOR: {alpha*100}%")
```

```
PEFORMING CLUSTERING FOR aggregation.txt
DESIRED NUMBER OF REP. POINTS: 8
DESIRED NUMBER OF CLUSTER: 7
SHRINKING FACTOR: 62.0%
```

In [7]:
```python
start = time.time()
data = data_set[:, 0:2]
Label_true = data_set[:, 2]

print(f"CLUSTERING {len(data)} POINTS  ... ")
print()

Label_pre = runCURE(data, numRepPoints, alpha, numDesCluster,
printClusterSize = False)
print(f"\nCOMPLETING CLUSTERING!")
print()

print(f"TIME TAKEN: {round(time.time() - start, 2)}s")
```

```
CLUSTERING 788 POINTS  ...

CURRENT CLUSTER COUNT: 700
CURRENT CLUSTER COUNT: 600
CURRENT CLUSTER COUNT: 500
CURRENT CLUSTER COUNT: 400
CURRENT CLUSTER COUNT: 300
CURRENT CLUSTER COUNT: 200
CURRENT CLUSTER COUNT: 100

FINAL CLUSTER COUNT: 7

COMPLETING CLUSTERING!

TIME TAKEN: 38.86s
```

In [8]:
```python
acc = v_measure_score(Label_true, Label_pre)
print(f"ACCURACY = {round(acc, 4)*100}%")
```

```
ACCURACY = 99.27%
```

In [9]:
```
plot(data_set, Label_pre)
```