```python
from hw4code.KMeans import KMeans,compute_purity,compute_NMI,getEuclideanDist
from hw4code.DataPoints import DataPoints
import random


class DBSCAN:
    # ------------------------------------------------------------------
    def __init__(self):
        self.e = 0.0
        self.minPts = 3
        self.noOfLabels = 0
    # ------------------------------------------------------------------
    def main(self, dataname):
        seed = 71

        self.dataname = dataname[5:-4]
        print("\nFor " + self.dataname)
        self.dataSet = KMeans.readDataSet(dataname)
        random.Random(seed).shuffle(self.dataSet)
        self.noOfLabels = DataPoints.getNoOFLabels(self.dataSet)
        self.e = self.getEpsilon(self.dataSet)
        print("Esp :" + str(self.e))
        self.dbscan(self.dataSet)


    # ------------------------------------------------------------------
    def getEpsilon(self, dataSet):
        distances = []
        sumOfDist = 0.0
        for i in range(len(dataSet)):
            point = dataSet[i]
            for j in range(len(dataSet)):
                if i == j:
                    continue
                pt = dataSet[j]
                dist = getEuclideanDist(point.x, point.y, pt.x, pt.y)
                distances.append(dist)

            distances.sort()
            sumOfDist += distances[7]
            distances = []
        return sumOfDist/len(dataSet)
    # ------------------------------------------------------------------
    def dbscan(self, dataSet):
        clusters = []
        visited = set()
        noise = set()

        # Iterate over data points
        for i in range(len(dataSet)):
            point = dataSet[i]
            if point in visited:
                continue
            visited.add(point)
            N = []
            minPtsNeighbours = 0

            # check which point satisfies minPts condition
            for j in range(len(dataSet)):
                if i==j:
```

```
                    continue
                pt = dataSet[j]
                dist = getEuclideanDist(point.x, point.y, pt.x, pt.y)
                if dist <= self.e:
                    minPtsNeighbours += 1
                    N.append(pt)

            if minPtsNeighbours >= self.minPts:
                cluster = set()
                cluster.add(point)
                point.isAssignedToCluster = True

                j = 0
                while j < len(N):
                    point1 = N[j]
                    minPtsNeighbours1 = 0
                    N1 = []
                    if not point1 in visited:
                        visited.add(point1)
                        for l in range(len(dataSet)):
                            pt = dataSet[l]
                            dist = getEuclideanDist(point1.x, point1.y, pt.x, pt.y)
                            if dist <= self.e:
                                minPtsNeighbours1 += 1
                                N1.append(pt)
                        if minPtsNeighbours1 >= self.minPts:
                            self.removeDuplicates(N, N1)

                    # Add point1 is not yet member of any other cluster then add it to cluster
                    # Hint: use self.isAssignedToCluster function to check if a point is assigned to any clusters
                    # ========================#
                    # STRART YOUR CODE HERE  #
                    # ========================#
                    if not point1.isAssignedToCluster:
                        cluster.add(point1)
                        point1.isAssignedToCluster = True
                    # ========================#
                    #   END YOUR CODE HERE   #
                    # ========================#
                    j += 1

                # add cluster to the list of clusters
                clusters.append(cluster)

            else:
                noise.add(point)


        # List clusters
        print("Number of clusters formed :" + str(len(clusters)))
        print("Noise points :" + str(len(noise)))

        # Calculate purity
        compute_purity(clusters,len(self.dataSet))
        compute_NMI(clusters,self.noOfLabels)
        DataPoints.writeToFile(noise, clusters, "DBSCAN_"+ self.dataname + ".csv")
    # ------------------------------------------------------------------
    def removeDuplicates(self, n, n1):
```

```
        for point in n1:
            isDup = False
            for point1 in n:
                if point1 == point:
                    isDup = True
                    break
            if not isDup:
                n.append(point)
```