```python
from hw4code.DataPoints import DataPoints
from collections import Counter
import random
import sys
import math
import pandas as pd
import numpy as np

# ========================================================================
def sqrt(n):
    return math.sqrt(n)

# ========================================================================
def getEuclideanDist(x1, y1, x2, y2):
    dist = sqrt(pow((x2 - x1), 2) + pow((y2 - y1), 2))
    return dist
# ========================================================================
def compute_purity(clusters,total_points):
    # Calculate purity

    # Create list to store the maximum union number for each output cluster.
    maxLabelCluster = []
    num_clusters = len(clusters)
    # ========================#
    # STRART YOUR CODE HERE  #
    # ========================#
    for i in range(num_clusters):
        cluster = clusters[i]
        original_labels = [x.label for x in cluster]
        max_union = max(original_labels, key=original_labels.count)
        maxLabelCluster.append(max_union)
    # ========================#
    #   END YOUR CODE HERE   #
    # ========================#
    purity = 0.0
    for j in range(num_clusters):
        purity += maxLabelCluster[j]
    purity /= total_points
    print("Purity is %.6f" % purity)

# ========================================================================
def compute_NMI(clusters,noOfLabels):
    # Get the NMI matrix first
    nmiMatrix = getNMIMatrix(clusters, noOfLabels)
    # Get the NMI matrix first
    nmi = calcNMI(nmiMatrix)
    print("NMI is %.6f" % nmi)


# ========================================================================
def getNMIMatrix(clusters, noOfLabels):
    # Matrix shape of [num_true_clusters + 1,num_output_clusters + 1] (example under week6's slide page 9)
    nmiMatrix = [[0 for x in range(len(clusters) + 1)] for y in range(noOfLabels + 1)]
    clusterNo = 0
    for cluster in clusters:
        # Create dictionary {true_class_No: Number of shared elements}
        labelCounts = {}
        # ========================#
```

```
        # STRART YOUR CODE HERE  #
        # ========================#
        labels = [x.label for x in cluster]
        labelCounts = Counter(labels)
        # ========================#
        #   END YOUR CODE HERE   #
        # ========================#
        labelTotal = 0
        labelCounts_sorted = sorted(labelCounts.items(), key=lambda item: item[1], reverse=True)
        for label, val in labelCounts_sorted:
            nmiMatrix[label - 1][clusterNo] = labelCounts[label]
            labelTotal += labelCounts.get(label)
        # Populate last row (row of summation)
        nmiMatrix[noOfLabels][clusterNo] = labelTotal
        clusterNo += 1
        labelCounts.clear()

    # Populate last col (col of summation)
    lastRowCol = 0
    for i in range(noOfLabels):
        totalRow = 0
        for j in range(len(clusters)):
            totalRow += nmiMatrix[i][j]
        lastRowCol += totalRow
        nmiMatrix[i][len(clusters)] = totalRow

    # Total number of datapoints
    nmiMatrix[noOfLabels][len(clusters)] = lastRowCol

    return nmiMatrix

# ================================================================================
def calcNMI(nmiMatrix):
    # Num of true clusters + 1
    row = len(nmiMatrix)
    # Num of output clusters + 1
    col = len(nmiMatrix[0])
    # Total number of datapoints
    N = nmiMatrix[row - 1][col - 1]
    I = 0.0
    HOmega = 0.0
    HC = 0.0

    for i in range(row - 1):
        for j in range(col - 1):
            # Compute the log part of each pair of clusters within I's formula.
            logPart_I = 1.0
            # ========================#
            # STRART YOUR CODE HERE  #
            # ========================#

            # stores total count in last row and column
            logPart_I = N * float(nmiMatrix[i][j]) / (float(nmiMatrix[i][-1]) * nmiMatrix[-1][j])

            # ========================#
            #   END YOUR CODE HERE   #
            # ========================#

```

```
117             if logPart_I == 0.0:
118                 continue
119             I += (nmiMatrix[i][j] / float(N)) * math.log(float(logPart_I))
120         # Compute HOmega
121         # ========================#
122         # STRART YOUR CODE HERE  #
123         # ========================#
124         p_wj = nmiMatrix[i][-1] / float(N)
125         HOmega += (p_wj * np.log(p_wj))
126         # ========================#
127         #   END YOUR CODE HERE   #
128         # ========================#
129 
130     #Compute HC
131     # ========================#
132     # STRART YOUR CODE HERE  #
133     # ========================#
134     for c in range(col-1):
135         p_cj = nmiMatrix[-1][c] / float(N)
136         HC += (p_cj * np.log(p_cj))
137     # ========================#
138     #   END YOUR CODE HERE   #
139     # ========================#
140 
141     return I / math.sqrt(HC * HOmega)
142 
143 
144 
145 
146 
147 # ============================================================================
148 class Centroid:
149     # ------------------------------------------------------------------------
150     def __init__(self, x, y):
151         self.x = x
152         self.y = y
153     # ------------------------------------------------------------------------
154     def __eq__(self, other):
155         if not type(other) is type(self):
156             return False
157         if other is self:
158             return True
159         if other is None:
160             return False
161         if self.x != other.x:
162             return False
163         if self.y != other.y:
164             return False
165         return True
166     # ------------------------------------------------------------------------
167     def __ne__(self, other):
168         result = self.__eq__(other)
169         if result is NotImplemented:
170             return result
171         return not result
172     # ------------------------------------------------------------------------
173     def toString(self):
174         return "Centroid [x=" + str(self.x) + ", y=" + str(self.y) + "]"
175     # ------------------------------------------------------------------------
176     def __str__(self):
```

```
        return self.toString()
    # ---------------------------------------------------------------------
    def __repr__(self):
        return self.toString()
```

```
# ===========================================================================
class KMeans:
    # ---------------------------------------------------------------------
    def __init__(self):
        self.K = 0
    # ---------------------------------------------------------------------
    def main(self, dataname,isevaluate=False):
        seed = 71
        self.dataname = dataname[5:-4]
        print("\nFor " + self.dataname)
        self.dataSet = self.readDataSet(dataname)
        self.K = DataPoints.getNoOFLabels(self.dataSet)
        random.Random(seed).shuffle(self.dataSet)
        self.kmeans(isevaluate)

    # ---------------------------------------------------------------------
    def check_dataloader(self,dataname):

        df = pd.read_table(dataname,sep = "\t", header=None, names=
['x','y','ground_truth_cluster'])
        print("\nFor " + dataname[5:-4] + ": number of datapoints is %d" %
df.shape[0])
        print(df.head(5))


    # ---------------------------------------------------------------------
    def kmeans(self,isevaluate=False):
        clusters = []
        k = 0
        while k < self.K:
            cluster = set()
            clusters.append(cluster)
            k += 1

        # Initially randomly assign points to clusters
        i = 0
        for point in self.dataSet:
            clusters[i % k].add(point)
            i += 1

        # calculate centroid for clusters
        centroids = []
        for j in range(self.K):
            centroids.append(self.getCentroid(clusters[j]))

        self.reassignClusters(self.dataSet, centroids, clusters)

        # continue till converge
        iteration = 0
```

```
        while True:
            iteration += 1
            # calculate centroid for clusters
            centroidsNew = []
            for j in range(self.K):
                centroidsNew.append(self.getCentroid(clusters[j]))

            isConverge = False
            for j in range(self.K):
                if centroidsNew[j] != centroids[j]:
                    isConverge = False
                else:
                    isConverge = True
            if isConverge:
                break

            for j in range(self.K):
                clusters[j] = set()

            self.reassignClusters(self.dataSet, centroidsNew, clusters)
            for j in range(self.K):
                centroids[j] = centroidsNew[j]
        print("Iteration :" + str(iteration))

        if isevaluate:
            # Calculate purity and NMI
            compute_purity(clusters, len(self.dataSet))
            compute_NMI(clusters, self.K)

        # write clusters to file for plotting
        f = open("Kmeans_"+ self.dataname + ".csv", "w")
        for w in range(self.K):
            print("Cluster " + str(w) + " size :" + str(len(clusters[w])))
            print(centroids[w].toString())
            for point in clusters[w]:
                f.write(str(point.x) + "," + str(point.y) + "," + str(w) + "\n")
        f.close()

    # ------------------------------------------------------------------
    def reassignClusters(self, dataSet, c, clusters):
        # reassign points based on cluster and continue till stable clusters found
        dist = [0.0 for x in range(self.K)]
        for point in dataSet:
            for i in range(self.K):
               dist[i] = getEuclideanDist(point.x, point.y, c[i].x, c[i].y)

            minIndex = self.getMin(dist)
            # assign point to the closest cluster
            # ========================#
            # STRART YOUR CODE HERE  #
            # ========================#
            clusters[minIndex].add(point)
            # ========================#
            #   END YOUR CODE HERE   #
            # ========================#
    # ------------------------------------------------------------------
    def getMin(self, dist):
        min = sys.maxsize
        minIndex = -1
        for i in range(len(dist)):
```

```
295             if dist[i] < min:
296                 min = dist[i]
297                 minIndex = i
298         return minIndex
299 
300     # ------------------------------------------------------------------
301     def getCentroid(self, cluster):
302         # mean of x and mean of y
303         cx = 0
304         cy = 0
305         # ========================#
306         # STRART YOUR CODE HERE  #
307         # ========================#
308         cx = np.average([c.x for c in cluster])
309         cy = np.average([c.y for c in cluster])
310         # ========================#
311         #   END YOUR CODE HERE   #
312         # ========================#
313         return Centroid(cx, cy)
314     # ------------------------------------------------------------------
315     @staticmethod
316     def readDataSet(filePath):
317         dataSet = []
318         with open(filePath) as f:
319             lines = f.readlines()
320         lines = [x.strip() for x in lines]
321         for line in lines:
322             points = line.split('\t')
323             x = float(points[0])
324             y = float(points[1])
325             label = int(points[2])
326             point = DataPoints(x, y, label)
327             dataSet.append(point)
328         return dataSet
329 
```