```python
from hw4code.DataPoints import DataPoints
import random
import sys
import math
import pandas as pd

# ============================================================================
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):
        labelCounts = {}
        for point in clusters[i]:
            if not point.label in labelCounts:
                labelCounts[point.label] = 0
            labelCounts[point.label] += 1
        max_union = -sys.maxsize - 1
        for label in labelCounts:
            if max_union < labelCounts[label]:
                max_union = labelCounts[label]
        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)]
```

```
    for cluster in clusters:
        # Create dictionary {true_class_No: Number of shared elements}
        labelCounts = {}
        # ======================#
        # STRART YOUR CODE HERE  #
        # ======================#
        for point in cluster:
            if not point.label in labelCounts:
                labelCounts[point.label] = 0
            labelCounts[point.label] += 1
        # ======================#
        #   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  #
            # ======================#
            logPart_I = (float(N) * nmiMatrix[i][j]) / (float(nmiMatrix[i]
```

```
116             # ========================#
117             #   END YOUR CODE HERE    #
118             # ========================#
119 
120             if logPart_I == 0.0:
121                 continue
122             I += (nmiMatrix[i][j] / float(N)) * math.log(float(logPart_I))
123         # Compute HOmega
124         # ========================#
125         # STRART YOUR CODE HERE  #
126         # ========================#
127             HOmega += nmiMatrix[row - 1][j]/float(N) * math.log(nmiMatrix[row
    - 1][j] / float(N))
128         # ========================#
129         #   END YOUR CODE HERE    #
130         # ========================#
131 
132     #Compute HC
133     # ========================#
134     # STRART YOUR CODE HERE  #
135     # ========================#
136         HC += nmiMatrix[i][col - 1]/float(N) * math.log(nmiMatrix[i][col -
    1]/float(N))
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     # ------------------------------------------------------------------------
```

```
174         return "Centroid [x=" + str(self.x) + ", y=" + str(self.y) + "]"
175     # ------------------------------------------------------------------
176     def __str__(self):
177         return self.toString()
178     # ------------------------------------------------------------------
179     def __repr__(self):
180         return self.toString()
181
182
183
184
185
186
187
188 # ============================================================================
189 class KMeans:
190     # ------------------------------------------------------------------
191     def __init__(self):
192         self.K = 0
193     # ------------------------------------------------------------------
194     def main(self, dataname,isevaluate=False):
195         seed = 71
196         self.dataname = dataname[5:-4]
197         print("\nFor " + self.dataname)
198         self.dataSet = self.readDataSet(dataname)
199         self.K = DataPoints.getNoOFLabels(self.dataSet)
200         random.Random(seed).shuffle(self.dataSet)
201         self.kmeans(isevaluate)
202
203     # ------------------------------------------------------------------
204     def check_dataloader(self,dataname):
205
206         df = pd.read_table(dataname,sep = "\t", header=None, names=
    ['x','y','ground_truth_cluster'])
207         print("\nFor " + dataname[5:-4] + ": number of datapoints is %d" %
    df.shape[0])
208         print(df.head(5))
209
210
211     # ------------------------------------------------------------------
212     def kmeans(self,isevaluate=False):
213         clusters = []
214         k = 0
215         while k < self.K:
216             cluster = set()
217             clusters.append(cluster)
218             k += 1
219
220         # Initially randomly assign points to clusters
221         i = 0
222         for point in self.dataSet:
223             clusters[i % k].add(point)
224             i += 1
225
226         # calculate centroid for clusters
227         centroids = []
228         for j in range(self.K):
229             centroids.append(self.getCentroid(clusters[j]))
230
```

```
232
233         # continue till converge
234         iteration = 0
235         while True:
236             iteration += 1
237             # calculate centroid for clusters
238             centroidsNew = []
239             for j in range(self.K):
240                 centroidsNew.append(self.getCentroid(clusters[j]))
241
242             isConverge = False
243             for j in range(self.K):
244                 if centroidsNew[j] != centroids[j]:
245                     isConverge = False
246                 else:
247                     isConverge = True
248             if isConverge:
249                 break
250
251             for j in range(self.K):
252                 clusters[j] = set()
253
254             self.reassignClusters(self.dataSet, centroidsNew, clusters)
255             for j in range(self.K):
256                 centroids[j] = centroidsNew[j]
257         print("Iteration :" + str(iteration))
258
259         if isevaluate:
260             # Calculate purity and NMI
261             compute_purity(clusters, len(self.dataSet))
262             compute_NMI(clusters, self.K)
263
264         # write clusters to file for plotting
265         f = open("Kmeans_"+ self.dataname + ".csv", "w")
266         for w in range(self.K):
267             print("Cluster " + str(w) + " size :" + str(len(clusters[w])))
268             print(centroids[w].toString())
269             for point in clusters[w]:
270                 f.write(str(point.x) + "," + str(point.y) + "," + str(w) +
"\n")
271         f.close()
272
273     # ------------------------------------------------------------------
274     def reassignClusters(self, dataSet, c, clusters):
275         # reassign points based on cluster and continue till stable clusters
found
276         dist = [0.0 for x in range(self.K)]
277         for point in dataSet:
278             for i in range(self.K):
279                dist[i] = getEuclideanDist(point.x, point.y, c[i].x, c[i].y)
280
281             minIndex = self.getMin(dist)
282             # assign point to the closest cluster
283             # ======================#
284             # STRART YOUR CODE HERE  #
285             # ======================#
286             for cluster in clusters:
287                 if point in cluster:
288                     cluster.remove(point)
```

```
290             # ========================#
291             #   END YOUR CODE HERE    #
292             # ========================#
293     # --------------------------------------------------------------------
294     def getMin(self, dist):
295         min = sys.maxsize
296         minIndex = -1
297         for i in range(len(dist)):
298             if dist[i] < min:
299                 min = dist[i]
300                 minIndex = i
301         return minIndex
302 
303     # --------------------------------------------------------------------
304     def getCentroid(self, cluster):
305         # mean of x and mean of y
306         cx = 0
307         cy = 0
308         # ========================#
309         # STRART YOUR CODE HERE  #
310         # ========================#
311         for data_point in cluster:
312             # print(data_point)
313             cx += data_point.x
314             cy += data_point.y
315         cx /= len(cluster)
316         cy /= len(cluster)
317         # ========================#
318         #   END YOUR CODE HERE    #
319         # ========================#
320         return Centroid(cx, cy)
321     # --------------------------------------------------------------------
322     @staticmethod
323     def readDataSet(filePath):
324         dataSet = []
325         with open(filePath) as f:
326             lines = f.readlines()
327         lines = [x.strip() for x in lines]
328         for line in lines:
329             points = line.split('\t')
330             x = float(points[0])
331             y = float(points[1])
332             label = int(points[2])
333             point = DataPoints(x, y, label)
334             dataSet.append(point)
335         return dataSet
336 
```