```python
from hw4code.DataPoints import DataPoints
from hw4code.KMeans import KMeans, compute_purity,compute_NMI
import math
from scipy.stats import multivariate_normal

# =============================================================================
class GMM:
    # -------------------------------------------------------------------------
    def __init__(self):
        self.dataSet = []
        self.K = 0
        self.mean = [[0.0 for x in range(2)] for y in range(3)]
        self.stdDev = [[0.0 for x in range(2)] for y in range(3)]
        self.coVariance = [[[0.0 for x in range(2)] for y in range(2)] for z in range(3)]
        self.W = None
        self.w = None
    # -------------------------------------------------------------------------
    def main(self, dataname):

        self.dataname = dataname[5:-4]
        print("\nFor " + self.dataname)
        self.dataSet = KMeans.readDataSet(dataname)
        self.K = DataPoints.getNoOFLabels(self.dataSet)
        # weight for pair of data and cluster
        self.W = [[0.0 for y in range(self.K)] for x in range(len(self.dataSet))]
        # weight for pair of data and cluster
        self.w = [0.0 for x in range(self.K)]
        self.GMM()

    # -------------------------------------------------------------------------
    def GMM(self):
        clusters = []
        # [num_clusters,2]
        self.mean = [[0.0 for y in range(2)] for x in range(self.K)]
        # [num_clusters,2]
        self.stdDev = [[0.0 for y in range(2)] for x in range(self.K)]
        # [num_clusters,2]
        self.coVariance = [[[0.0 for z in range(2)] for y in range(2)] for x in range(self.K)]
        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 % self.K].add(point)
            i += 1

        # Initially assign equal prior weight for each cluster
        for m in range(self.K):
            self.w[m] = 1.0 / self.K

        # Get Initial mean, std, covariance matrix
        DataPoints.getMean(clusters, self.mean)
```

```
        DataPoints.getCovariance(clusters, self.mean, self.stdDev, self.coVariance)

        length = 0
        while True:
            mle_old = self.Likelihood()
            self.Estep()
            self.Mstep()
            length += 1
            mle_new = self.Likelihood()

            # convergence condition
            if abs(mle_new - mle_old) / abs(mle_old) < 0.000001:
                break

        print("Number of Iterations = " + str(length))
        print("\nAfter Calculations")
        print("Final mean = ")
        self.printArray(self.mean)
        print("\nFinal covariance = ")
        self.print3D(self.coVariance)

        # Assign points to cluster depending on max prob.
        for j in range(self.K):
            clusters[j] = set()

        i = 0
        for point in self.dataSet:
            index = -1
            prob = 0.0
            for j in range(self.K):
                if self.W[i][j] > prob:
                    index = j
                    prob = self.W[i][j]
            temp = clusters[index]
            temp.add(point)
            i += 1

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

        # write clusters to file for plotting
        f = open("GMM_" + self.dataname + ".csv", "w")
        for w in range(self.K):
            print("Cluster " + str(w) + " size :" + str(len(clusters[w])))
            for point in clusters[w]:
                f.write(str(point.x) + "," + str(point.y) + "," + str(w) + "\n")
        f.close()
    # ------------------------------------------------------------------
    def Estep(self):
        # Update self.W
        for i in range(len(self.dataSet)):
            denominator = 0.0
            for j in range(self.K):
                gaussian = multivariate_normal(self.mean[j], self.coVariance[j])
                # Compute numerator for self.W[i][j] below
```

```
115                 # ========================#
116                 # STRART YOUR CODE HERE  #
117                 # ========================#
118                 numerator = self.w[j] * gaussian.pdf([self.dataSet[i].x,
    self.dataSet[i].y])
119                 # ========================#
120                 #   END YOUR CODE HERE   #
121                 # ========================#
122                 self.W[i][j] = numerator
123                 denominator += numerator
124 
125             # normalize W[i][j] into probabilities
126             # ========================#
127             # STRART YOUR CODE HERE  #
128             # ========================#
129             for j in range(self.K):
130                 self.W[i][j] /= denominator
131             # ========================#
132             #   END YOUR CODE HERE   #
133             # ========================#
134     # ------------------------------------------------------------------
135     def Mstep(self):
136         for j in range(self.K):
137             denominator = 0.0
138             numerator_x = 0.0
139             numerator_y = 0.0
140             cov_xy = 0.0
141             updatedMean_x = 0.0
142             updatedMean_y = 0.0
143 
144             # update self.w[j] and self.mean
145             for i in range(len(self.dataSet)):
146                 denominator += self.W[i][j]
147                 updatedMean_x += self.W[i][j] * self.dataSet[i].x
148                 updatedMean_y += self.W[i][j] * self.dataSet[i].y
149 
150             self.w[j] = denominator / len(self.dataSet)
151 
152             #update self.mean
153             # ========================#
154             # STRART YOUR CODE HERE  #
155             # ========================#
156             self.mean[j][0] = updatedMean_x / denominator
157             self.mean[j][1] = updatedMean_y / denominator
158             # ========================#
159             #   END YOUR CODE HERE   #
160             # ========================#
161 
162             # update covariance matrix
163             for i in range(len(self.dataSet)):
164                 numerator_x += self.W[i][j] * pow((self.dataSet[i].x -
    self.mean[j][0]), 2)
165                 numerator_y += self.W[i][j] * pow((self.dataSet[i].y -
    self.mean[j][1]), 2)
166                 # Compute conv_xy +=?
167                 # ========================#
168                 # STRART YOUR CODE HERE  #
169                 # ========================#
170                 cov_xy += self.W[i][j] * (self.dataSet[i].x - self.mean[j]
```

```
171                 # ========================#
172                 #   END YOUR CODE HERE    #
173                 # ========================#
174 
175             self.stdDev[j][0] = numerator_x / denominator
176             self.stdDev[j][1] = numerator_y / denominator
177 
178 
179             self.coVariance[j][0][0] = self.stdDev[j][0]
180             self.coVariance[j][1][1] = self.stdDev[j][1]
181             self.coVariance[j][0][1] = self.coVariance[j][1][0] = cov_xy /
    denominator
182     # ------------------------------------------------------------------
183     def Likelihood(self):
184         likelihood = 0.0
185         for i in range(len(self.dataSet)):
186             numerator = 0.0
187             for j in range(self.K):
188                 gaussian = multivariate_normal(self.mean[j],
    self.coVariance[j])
189                 numerator += self.w[j] * gaussian.pdf([self.dataSet[i].x,
    self.dataSet[i].y])
190             likelihood += math.log(numerator)
191         return likelihood
192     # ------------------------------------------------------------------
193     def printArray(self, mat):
194         for i in range(len(mat)):
195             for j in range(len(mat[i])):
196                 print(str(mat[i][j]) + " "),
197             print("")
198     # ------------------------------------------------------------------
199     def print3D(self, mat):
200         for i in range(len(mat)):
201             print("For Cluster : " + str((i + 1)))
202             for j in range(len(mat[i])):
203                 for k in range(len(mat[i][j])):
204                     print(str(mat[i][j][k]) + " "),
205                 print("")
206             print("")
207 
208 # ======================================================================
209 if __name__ == "__main__":
210     g = GMM()
211     dataname = "dataset1.txt"
212     g.main(dataname)
```