CS145 Howework 4

Important Note: HW4 is due on **11:59 PM PT, Nov 20 (Friday, Week 7)**. Please submit through GradeScope.

Print Out Your Name and UID

Name: Rui Deng, UID: 205123245

Before You Start

You need to first create HW4 conda environment by the given `cs145hw4.yml` file, which provides the name and necessary packages for this tasks. If you have `conda` properly installed, you may create, activate or deactivate by the following commands:

```
conda env create -f cs145hw4.yml
conda activate hw4
conda deactivate
```

OR

```
conda env create --name NAMEOFYOURCHOICE -f cs145hw4.yml
conda activate NAMEOFYOURCHOICE
conda deactivate
```

To view the list of your environments, use the following command:

```
conda env list
```

More useful information about managing environments can be found here (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html).

You may also quickly review the usage of basic Python and Numpy package, if needed in coding for matrix operations.

In this notebook, you must not delete any code cells in this notebook. If you change any code outside the blocks (such as some important hyperparameters) that you are allowed to edit (between STRART/END YOUR CODE HERE), you need to highlight these changes. You may add some additional cells to help explain your results and observations.

In [1]:

```
import numpy as np
import pandas as pd
import sys
import random
import math
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
%load_ext autoreload
%autoreload 2
```

If you can successfully run the code above, there will be no problem for environment setting.

1. Clustering Evaluation

This workbook will walk you through an example for calculating different clustering metrics.

Note: This is a "question-answer" style problem. You do not need to code anything and you are required to calculate by hand (with a scientific calculator).

Questions

Suppose we want to cluster the following 20 conferences into four areas, with ground truth label and algorithm output label shown in third and fourth column. Please evaluate the quality of the clustering algorithm according to four different metrics respectively.

Questions (please include intermediate steps)

1. Calculate purity.
2. Calculate precision.
3. Calculate recall.
4. Calculate F1-score.
5. Calculate normalized mutual information.

Your answer here:

Note: you can use several code cells to help you compute the results and answer the questions. Again you don't need to do any coding.

Please type your answer here!

answer 1

We assign algorithm output label 1 to ground truth label 2, and all of the 5 datapoints are matched.

We assign algorithm output label 2 to ground truth label 3, and 5 out of the 6 datapoints are matched. (the non-matched datapoint has ground truth label 1)

We assign algorithm output label 3 to ground truth label 1, and 4 out of the 5 datapoints are matched. (the non-matched datapoint has ground truth label 4)

We assign algorithm output label 4 to ground truth label 4, and 4 out of the 4 datapoints are matched.

So the purity is:

$$purity = \frac{1}{N}\sum_{k} max|c_k \cap \omega_j| = \frac{1}{20} * (5+5+4+4) = 0.9$$

answer 2

Precision is $\frac{TP}{TP+FP}$. Then we use coding to calculate TP, FP, TN, and FN:

In [4]:

```
ground_truth = [3, 3, 1, 1, 1, 4, 3, 3, 4, 2, 4, 2, 1, 2, 3, 2, 1, 2, 4, 4]
output_label = [2, 2, 3, 3, 3, 4, 2, 2, 3, 1, 4, 1, 3, 1, 2, 1, 2, 1, 4, 4]
TP = 0.0
FP = 0.0
TN = 0.0
FN = 0.0
for i in range(len(ground_truth)):
    for j in range(i + 1, len(ground_truth)):
        g = ground_truth[i]
        l = output_label[j]
        if output_label[i] == output_label[j]:
            if ground_truth[i] == ground_truth[j]:
                TP += 1
            else:
                FP += 1
        else:
            if ground_truth[i] == ground_truth[j]:
                FN += 1
            else:
                TN += 1
precision = TP / (TP + FP)
print(precision)
```

```
0.7804878048780488
```

So the precision is 0.78.

answer 3

Recall is $\frac{TP}{TP+FN}$. Using the previous code, we have:

In [5]:

```
recall = TP / (TP + FN)
print(recall)
```

```
0.8
```

So the recall is 0.8.

answer 4

F1-score is $\frac{2*Precision*Recall}{Precision+Recall}$. Using the previous code, we have:

In [6]:

```
print(2 * precision * recall / (precision + recall))
```

```
0.7901234567901235
```

So the F-1 score is 0.79.

answer 5

We use coding to generate normalized mutual information:

In [13]:

```
from collections import Counter
import math

N = 20.0
dic_output = Counter(output_label)
dic_truth = Counter(ground_truth)
H_O = 0.0
H_T = 0.0
for i in range(1, 5):
    H_O -= dic_output[i] / N * math.log(dic_output[i] / N, 2)
    H_T -= dic_truth[i] / N * math.log(dic_truth[i] / N, 2)
I = 0.0
I += 5 / N * math.log(N * 5 / (dic_output[1] * dic_truth[2]), 2)
I += 5 / N * math.log(N * 5 / (dic_output[2] * dic_truth[3]), 2) + 1 / N * math.
log(N * 1 / (dic_output[2] * dic_truth[1]), 2)
I += 4 / N * math.log(N * 4 / (dic_output[3] * dic_truth[1]), 2) + 1 / N * math.
log(N * 1 / (dic_output[3] * dic_truth[4]), 2)
I += 4 / N * math.log(N * 4 / (dic_output[4] * dic_truth[4]), 2)
print(I / (H_O * H_T) ** (1 / 2))
```

```
0.8152212305376372
```

So the NMI is 0.8152.

2. K-means

In this section, we are going to apply K-means algorithm against two datasets (dataset1.txt, dataset2.txt) with different distributions, respectively.

For each dataset, it contains 3 columns, with the format: x1 \t x2 \t cluster_label. You need to use the first two columns for clustering, and the last column for evaluation.

In [14]:

```
from hw4code.KMeans import KMeans
k = KMeans()
# As a sanity check, we print out a sample of each dataset
dataname1 = "data/dataset1.txt"
dataname2 = "data/dataset2.txt"
k.check_dataloader(dataname1)
k.check_dataloader(dataname2)
```

```
For dataset1: number of datapoints is 150
          x         y  ground_truth_cluster
0 -0.163880 -0.219869                     1
1 -0.886274 -0.356186                     1
2 -0.978910 -0.893314                     1
3 -0.658867 -0.371122                     1
4 -0.072518  0.399157                     1

For dataset2: number of datapoints is 200
          x         y  ground_truth_cluster
0  1.068587  0.136921                     1
1  0.705440  0.393068                     1
2  0.840811 -0.054906                     1
3 -0.923447  0.598501                     1
4  0.784353  0.724743                     1
```

2.1 Coding K-means

Complete the `reassignClusters` and `getCentroid` function in `KMeans.py` .

Print out each output cluster's size and centroid （x,y) for dataset1 and dataset2 respectively.

In [17]:

```
k = KMeans()
#=======================#
# STRART YOUR CODE HERE  #
#=======================#
k.main("data/dataset1.txt")
k.main("data/dataset2.txt")
#=======================#
#   END YOUR CODE HERE   #
#=======================#
```

```
For dataset1
Iteration :4
Cluster 0 size :50
Centroid [x=2.5737264423871213, y=-0.027462568841232993]
Cluster 1 size :50
Centroid [x=-0.4633368646347212, y=-0.46611409698195794]
Cluster 2 size :50
Centroid [x=0.9888766205736857, y=2.010478965197201]

For dataset2
Iteration :3
Cluster 0 size :102
Centroid [x=1.2708406269481844, y=-0.08583389704900128]
Cluster 1 size :98
Centroid [x=-0.2018593506236788, y=0.5726963240559535]
```

2.2 Purity and NMI Evaluation

Complete the `compute_purity` function in `KMeans.py` .

In order to compute NMI, you need to firstly compute NMI matrix and then do the calculation. That is to complete the `getNMIMatrix` and `calcNMI` functions in `KMeans.py` .

Print out the purity and NMI for each dataset respectively.

In [24]:

```
k = KMeans()
#=======================#
# STRART YOUR CODE HERE  #
#=======================#
k.main("data/dataset1.txt", isevaluate=True)
k.main("data/dataset2.txt", isevaluate=True)
#=======================#
#   END YOUR CODE HERE   #
#=======================#
```

```
For dataset1
Iteration :4
Purity is 1.000000
NMI is 1.000000
Cluster 0 size :50
Centroid [x=2.5737264423871213, y=-0.027462568841232993]
Cluster 1 size :50
Centroid [x=-0.4633368646347212, y=-0.46611409698195794]
Cluster 2 size :50
Centroid [x=0.9888766205736857, y=2.010478965197201]

For dataset2
Iteration :3
Purity is 0.760000
NMI is 0.205096
Cluster 0 size :102
Centroid [x=1.2708406269481844, y=-0.08583389704900128]
Cluster 1 size :98
Centroid [x=-0.2018593506236788, y=0.5726963240559535]
```

2.3 Visualization

The clustering results for KMeans are saved as `KMeans_dataset1.csv` and `KMeans_dataset2.csv` respectively under your root folder. Plot the clustering results for the two datasets, with different colors representing different clusters.

In [41]:

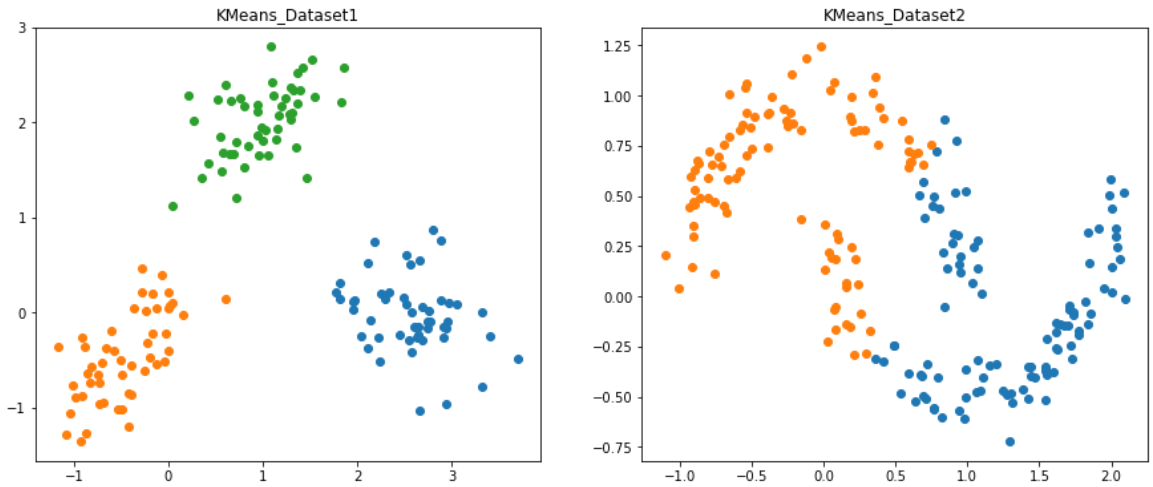
```
CSV_FILE_PATH1 = 'Kmeans_dataset1.csv'
CSV_FILE_PATH2 = 'Kmeans_dataset2.csv'

df1 = pd.read_csv(CSV_FILE_PATH1,header=None,names=['x','y','pred'])
df2 = pd.read_csv(CSV_FILE_PATH2,header=None,names=['x','y','pred'])
fig, [ax0,ax1] = plt.subplots(1, 2, figsize=(15, 6))
ax0.title.set_text("KMeans_Dataset1")
ax1.title.set_text("KMeans_Dataset2")

#========================#
# STRART YOUR CODE HERE  #
#========================#
def create_plot(df, ax):
    df_0 = df.loc[df.iloc[:, 2]==0]
    df_1 = df.loc[df.iloc[:, 2]==1]
    df_2 = df.loc[df.iloc[:, 2]==2]
    df_3 = df.loc[df.iloc[:, 2]==3]
    ax.scatter(df_0.iloc[:, 0], df_0.iloc[:, 1])
    ax.scatter(df_1.iloc[:, 0], df_1.iloc[:, 1])
    ax.scatter(df_2.iloc[:, 0], df_2.iloc[:, 1])
    ax.scatter(df_3.iloc[:, 0], df_3.iloc[:, 1])

create_plot(df1, ax0)
create_plot(df2, ax1)

#========================#
#   END YOUR CODE HERE   #
#========================#
plt.show()
```



Question

Give the pros and cons of K-means algorithm. (At least one for pro and two for cons to get full marks)

Your answer here

Please type your answer here!

Pros:

This algorithm is very efficient, and its complexity is linear in n (number of datapoints). Specifically, it has O(ktn), where k is number of clusters and t is the number of iterations, and k, t << n.

Cons:

1. This algorithm is not suitable to discover clusters with non-convex shape; for example, in dataset2, k-Means cannot efficiently classify the two curved clusters.

2. Since we use the mean as center, this algorithm is sensitive to outliers and noisy data.

3. The algorithm requires us to set k (number of clusters) at first.

3 DBSCAN

In this section, we are going to use DBSCAN for clustering the same two datasets.

3.1 Coding DBSCAN

Complete the `dbscan` function in `DBSCAN.py` . Print out the purity, NMI and cluter size for each dataset respectively.

In [39]:

```
from hw4code.DBSCAN import DBSCAN
d = DBSCAN()
#========================#
# STRART YOUR CODE HERE  #
#========================#
d.main("data/dataset1.txt")
d.main("data/dataset2.txt")
#========================#
#   END YOUR CODE HERE   #
#========================#
```

```
For dataset1
Esp :0.3560832705047313
Number of clusters formed :4
Noise points :11
Purity is 0.940000
NMI is 0.959065
Cluster 0 size :49
Cluster 1 size :41
Cluster 2 size :47
Cluster 3 size :4

For dataset2
Esp :0.18652096476712493
Number of clusters formed :3
Noise points :3
Purity is 0.985000
NMI is 0.817349
Cluster 0 size :99
Cluster 1 size :51
Cluster 2 size :47
```

3.2 Visualization

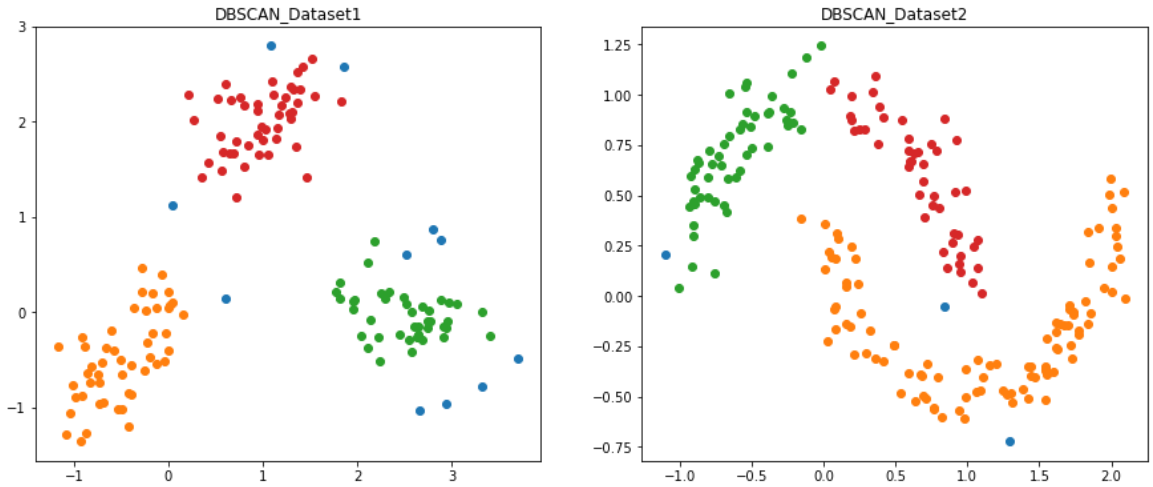
The clustering results for DBSCAN are saved as `DBSCAN_dataset1.csv` and `DBSCAN_dataset2.csv` respectively under your root folder. Plot the clustering results for the two datasets, with different colors representing different clusters.

In [42]:

```
CSV_FILE_PATH1 = 'DBSCAN_dataset1.csv'
CSV_FILE_PATH2 = 'DBSCAN_dataset2.csv'

df1 = pd.read_csv(CSV_FILE_PATH1,header=None,names=['x','y','pred'])
df2 = pd.read_csv(CSV_FILE_PATH2,header=None,names=['x','y','pred'])
fig, [ax0,ax1] = plt.subplots(1, 2, figsize=(15, 6))
ax0.title.set_text("DBSCAN_Dataset1")
ax1.title.set_text("DBSCAN_Dataset2")

#========================#
# STRART YOUR CODE HERE  #
#========================#
create_plot(df1, ax0)
create_plot(df2, ax1)
#========================#
#   END YOUR CODE HERE   #
#========================#
plt.show()
```



Question

Give the pros and cons of DBSCAN algorithm. (At least two for pro and one for cons to get full marks)

Your answer here

Please type your answer here!

Pros:

1. It can efficiently detect and identify the noise/outliers in data, so the algorithm is robust to noises.

2. It can discover non-convex or even arbitrary shape in clusters, as displayed in the classification result of dataset2.

Cons:

This algorithm fails if clusters have varying density, since we have to set the minPoints as a hyperparameter before the algorithm starts.

4 GMM

In this section, we are going to use GMM for clustering the same two datasets.

4.1 Coding GMM

Complete the `Estep` and 'Mstep' function in `GMM.py` . Print out the purity, NMI, final mean, covariance and cluter size for each dataset respectively.

In [47]:

```
from hw4code.GMM import GMM
g = GMM()
#========================#
# STRART YOUR CODE HERE  #
#========================#
g.main("data/dataset1.txt")
g.main("data/dataset2.txt")
#========================#
#   END YOUR CODE HERE   #
#========================#
```

```
For dataset1
Number of Iterations = 22

After Calculations
Final mean =
-0.46247285694404044
-0.4638749980764899

0.9898929396029765
2.011802723814242

2.57342634413319
-0.027108746076609493


Final covariance =
For Cluster : 1
0.14918910487220216
0.1173463005433889

0.1173463005433889
0.21554861253107502


For Cluster : 2
0.16028233507625483
0.07486967581052754

0.07486967581052754
0.13939774162738802


For Cluster : 3
0.18039223672749394
-0.04672614559811056

-0.04672614559811056
0.15206459963738583


Purity is 1.000000
NMI is 1.000000
Cluster 0 size :50
Cluster 1 size :50
Cluster 2 size :50

For dataset2
Number of Iterations = 95

After Calculations
Final mean =
0.7464905663922623
0.4564966584854107

0.28287851889390975
-0.05970560727188754


Final covariance =
For Cluster : 1
0.769279076535834
```

```
-0.28782809642382134

-0.28782809642382134
0.1901249384356509


For Cluster : 2
0.6828574757628691
-0.300589159943905

-0.300589159943905
0.17583559485120043


Purity is 0.690000
NMI is 0.107406
Cluster 0 size :106
Cluster 1 size :94
```

4.2 Visualization

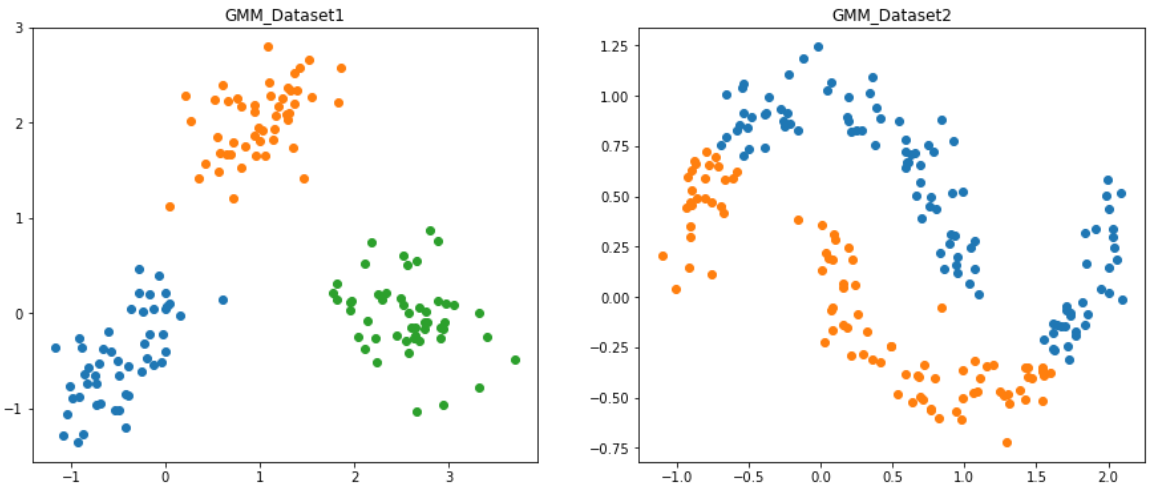
The clustering results for GMM are saved as `GMM_dataset1.csv` and `GMM_dataset2.csv` respectively under your root folder. Plot the clustering results for the two datasets, with different colors representing different clusters.

In [48]:

```
CSV_FILE_PATH1 = 'GMM_dataset1.csv'
CSV_FILE_PATH2 = 'GMM_dataset2.csv'

df1 = pd.read_csv(CSV_FILE_PATH1,header=None,names=['x','y','pred'])
df2 = pd.read_csv(CSV_FILE_PATH2,header=None,names=['x','y','pred'])
fig, [ax0,ax1] = plt.subplots(1, 2, figsize=(15, 6))
ax0.title.set_text("GMM_Dataset1")
ax1.title.set_text("GMM_Dataset2")

#========================#
# STRART YOUR CODE HERE  #
#========================#
create_plot(df1, ax0)
create_plot(df2, ax1)
#========================#
#   END YOUR CODE HERE   #
#========================#
plt.show()
```



Questions

1. Give the pros and cons of GMM algorithm. (At least two for pro and two for cons to get full marks)
2. Compare the visualization results from three algorithms, analyze for each dataset why these algorithms would produce such result.

Your answer here:

Please type your answer here!

Pros of GMM:

1. GMM can efficiently classify clusters with different sizes and density

2. GMM do not require us the set k (number of clusters) before the algorithm starts.

3. GMM is a generative model; that is to say, we can generate any new datapoint given its cluster distribution.

Cons of GMM:

1. It is not suitable for clusters with non-convex shape, such as dataset2.

2. It has high computational cost if the dimensions of datapoints and the number of datapoints are high.

Reasoning over dataset1:

We can see that both k-Means and GMM have the perfect purity and NMI classification result. For k-Means, we set the correct k (i.e. number of clusters) initially; for GMM, we run the algorithm long-enough for it to converge. Under such conditions, both algorithms can successfully detect the convex shaped cluster in dataset1.

However, for DBSCAN, it only has a purity of 0.94 and NMI of 0.959. This is because that for the left-most cluster, it is more sparsely distributed comparing to others. DBSCAN is less efficient when classifying clusters with different density, so it classifies points at the margin of the left-most cluster as outliers, and thus results in a lower purity.

Reasoning over dataset2:

This dataset has a non-convex shape. As indicated previously, k-Means and GMM are not suitable for non-convex cluster classification. As a result, k-Means has a purity of 0.76 and NMI of 0.205; GMM only has a purity of 0.69 and NMI of 0.1074.

On the contrary, DBSCAN can efficiently detect clusters with non-convex shape. So in this case, it has purity of 0.985 and NMI of 0.817. Moreover, different from dataset1, these two non-convex clusters has similar densities, so DBSCAN only classifies 3 marginal points as outliers and has higher accuracy.

5 Bonus Question

Prove that KMeans algorithm would guarantee covergence. (**Hint: prove for each step the loss would descrease.**)

Please type your answer here!

End of Homework 4 :)

After you've finished the homework, please print out the entire `ipynb` notebook and four `py` files into one PDF file. Make sure you include the output of code cells and answers for questions. Prepare submit it to GradeScope. Also this time remember assign the pages to the questions on GradeScope

```
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 cluster in clusters:
        l = [val.label for val in cluster]
        label = max(set(l), key=l.count)
        maxLabelCluster.append(sum([l_i == label for l_i in l]))
    # ========================#
    #   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  #
        # ========================#
        for point in cluster:
            if point.label not in labelCounts:
                labelCounts[point.label] = 1
            else:
                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] / (nmiMatrix[i][col - 1] *
             nmiMatrix[row - 1][j])
            # ========================#
            #   END YOUR CODE HERE   #
            # ========================#

            if logPart_I == 0.0:
                continue
            I += (nmiMatrix[i][j] / float(N)) * math.log(float(logPart_I))
        # Compute HOmega
        # ========================#
        # STRART YOUR CODE HERE  #
        # ========================#
        w_i = nmiMatrix[i][col - 1]
        if w_i != 0:
            HOmega -= w_i / float(N) * math.log(w_i / float(N))
        # ========================#
        #   END YOUR CODE HERE   #
        # ========================#

    #Compute HC
    # ========================#
    # STRART YOUR CODE HERE  #
    # ========================#
    for j in range(col - 1):
        c_j = nmiMatrix[row - 1][j]
        if c_j != 0:
            HC -= c_j / float(N) * math.log(c_j / float(N))
    # ========================#
    #   END YOUR CODE HERE   #
    # ========================#

    return I / math.sqrt(HC * HOmega)




# ============================================================================
class Centroid:
    # ----------------------------------------------------------------------
```

```
    def __init__(self, x, y):
        self.x = x
        self.y = y
    # ------------------------------------------------------------------
    def __eq__(self, other):
        if not type(other) is type(self):
            return False
        if other is self:
            return True
        if other is None:
            return False
        if self.x != other.x:
            return False
        if self.y != other.y:
            return False
        return True
    # ------------------------------------------------------------------
    def __ne__(self, other):
        result = self.__eq__(other)
        if result is NotImplemented:
            return result
        return not result
    # ------------------------------------------------------------------
    def toString(self):
        return "Centroid [x=" + str(self.x) + ", y=" + str(self.y) + "]"
    # ------------------------------------------------------------------
    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)):
        if dist[i] < min:
            min = dist[i]
            minIndex = i
    return minIndex
```

```
# ---------------------------------------------------------------------
def getCentroid(self, cluster):
    # mean of x and mean of y
    cx = 0
    cy = 0
    # ========================#
    # STRART YOUR CODE HERE  #
    # ========================#
    cx = sum([val.x for val in cluster]) / len(cluster)
    cy = sum([val.y for val in cluster]) / len(cluster)
    # ========================#
    #   END YOUR CODE HERE   #
    # ========================#
    return Centroid(cx, cy)
# ---------------------------------------------------------------------
@staticmethod
def readDataSet(filePath):
    dataSet = []
    with open(filePath) as f:
        lines = f.readlines()
    lines = [x.strip() for x in lines]
    for line in lines:
        points = line.split('\t')
        x = float(points[0])
        y = float(points[1])
        label = int(points[2])
        point = DataPoints(x, y, label)
        dataSet.append(point)
    return dataSet
```

```
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  #
        # ========================#
        def isAssignedToCluster(point, clusters):
            for cluster in clusters:
                for pt in cluster:
                    if pt.x == point.x and pt.y == point.y:
                        return True
            return False
```

```
                if not isAssignedToCluster(point1, clusters):
                    cluster.add(point1)
                # ========================#
                #   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)
```

```
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.getStdDeviation(clusters, self.mean, self.stdDev)
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
            numerator = 0.0
            # ========================#
            # STRART YOUR CODE HERE  #
            # ========================#
            xi = [self.dataSet[i].x, self.dataSet[i].y]
            numerator = self.w[j] * gaussian.pdf(xi)
            # ========================#
            #   END YOUR CODE HERE   #
            # ========================#
            self.W[i][j] = numerator
            denominator += numerator

        # normalize W[i][j] into probabilities
        # ========================#
        # STRART YOUR CODE HERE  #
        # ========================#
        for j in range(self.K):
            self.W[i][j] /= denominator
        # ========================#
        #   END YOUR CODE HERE   #
        # ========================#
# ------------------------------------------------------------------
def Mstep(self):
    for j in range(self.K):
        denominator = 0.0
        numerator_x = 0.0
        numerator_y = 0.0
        cov_xy = 0.0
        updatedMean_x = 0.0
        updatedMean_y = 0.0

        # update self.w[j] and self.mean
        for i in range(len(self.dataSet)):
            denominator += self.W[i][j]
```

```
            updatedMean_x += self.W[i][j] * self.dataSet[i].x
            updatedMean_y += self.W[i][j] * self.dataSet[i].y

        self.w[j] = denominator / len(self.dataSet)

        #update self.mean
        # ========================#
        # STRART YOUR CODE HERE  #
        # ========================#
        self.mean[j][0] = updatedMean_x / denominator
        self.mean[j][1] = updatedMean_y / denominator
        # ========================#
        #   END YOUR CODE HERE   #
        # ========================#

        # update covariance matrix
        for i in range(len(self.dataSet)):
            numerator_x += self.W[i][j] * pow((self.dataSet[i].x -
             self.mean[j][0]), 2)
            numerator_y += self.W[i][j] * pow((self.dataSet[i].y -
             self.mean[j][1]), 2)
            # Compute conv_xy +=?
            # ========================#
            # STRART YOUR CODE HERE  #
            # ========================#
            cov_xy += self.W[i][j] * (self.dataSet[i].x - self.mean[j][0])
             * (self.dataSet[i].y - self.mean[j][1])
            # ========================#
            #   END YOUR CODE HERE   #
            # ========================#

        self.stdDev[j][0] = numerator_x / denominator
        self.stdDev[j][1] = numerator_y / denominator


        self.coVariance[j][0][0] = self.stdDev[j][0]
        self.coVariance[j][1][1] = self.stdDev[j][1]
        self.coVariance[j][0][1] = self.coVariance[j][1][0] = cov_xy /
         denominator
# ----------------------------------------------------------------------
def Likelihood(self):
    likelihood = 0.0
    for i in range(len(self.dataSet)):
        numerator = 0.0
        for j in range(self.K):
            gaussian = multivariate_normal(self.mean[j],
             self.coVariance[j])
            numerator += self.w[j] * gaussian.pdf([self.dataSet[i].x,
             self.dataSet[i].y])
        likelihood += math.log(numerator)
    return likelihood
```

```
    # ------------------------------------------------------------------
    def printArray(self, mat):
        for i in range(len(mat)):
            for j in range(len(mat[i])):
                print(str(mat[i][j]) + " "),
            print("")
    # ------------------------------------------------------------------
    def print3D(self, mat):
        for i in range(len(mat)):
            print("For Cluster : " + str((i + 1)))
            for j in range(len(mat[i])):
                for k in range(len(mat[i][j])):
                    print(str(mat[i][j][k]) + " "),
                print("")
            print("")

# ======================================================================
if __name__ == "__main__":
    g = GMM()
    dataname = "dataset1.txt"
    g.main(dataname)
```