```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)):
```

```
                    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)
                        noise.discard(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 +
```

```
    # ------------------------------------------------------------------
    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)


```

```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)
```

```python
        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
```

```python
                # ========================#
                # STRART YOUR CODE HERE  #
                # ========================#
                numerator = self.w[j] * gaussian.pdf([self.dataSet[i].x, self.dataSet[i].y])
                # ========================#
                #   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]
```

```
                # ========================#
                #   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)
```

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: Ali Mirabzadeh, UID: 305179067

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)](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.

ID	Conference Name	Ground Truth Label	Algorithm output Label
1	IJCAI	3	2
2	AAAI	3	2
3	ICDE	1	3
4	VLDB	1	3
5	SIGMOD	1	3
6	SIGIR	4	4
7	ICML	3	2
8	NIPS	3	2
9	CIKM	4	3
10	KDD	2	1
11	WWW	4	4
12	PAKDD	2	1
13	PODS	1	3
14	ICDM	2	1
15	ECML	3	2
16	PKDD	2	1
17	EDBT	1	2
18	SDM	2	1
19	ECIR	4	4
20	WSDM	4	4

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

Cluster	Truth	Name	Cluster	Output label	Name
w_1	3	IJCAI	C_1	2	IJCAI
	3	AAAI		2	AAAI
	3	ICML		2	ICML
	3	NIPS		2	NIPS
	3	ECML		2	EDBT
w_2	1	ICDE		2	ECML
	1	SIGMOD	C_2	3	ICDE
	1	PODS		3	VLDB
	1	EDBT		3	SIGMOD
	1	VLDB		3	CIKM
w_3	2	KDD		3	PODS
	2	PAKDD	C_3	4	SIGIR
	2	ICDM		4	WWW
	2	PKDD		4	ECIR
	2	SDM		4	WSDM
w_4	4	SIGIR	C_4	1	KDD
	4	CIKM		1	PAKDD
	4	WWW		1	ICDM
	4	ECIR		1	PKDD
	4	WSDM		1	SDM

① $C_k = \{C_1, C_2, C_3, C_4\}$, $w_j = \{w_1, w_2, w_3, w_4\}$

$C_1 \to w_3$: majority 5, $C_2 \to w_1$: majority : 5

$C_3 \to w_4$: majority 4, $C_4 \to w_2$: majority : 5

$$\text{Purity} = \frac{5+5+5+4}{20} = \frac{19}{20} = \underline{0.95}$$

answer 2, 3, 4

- Random Index (RI) = (TP+TN)/(TP+FP+FN+TN)
- F-measure: 2Precision*Recall/(Precision+Recall)
  - Precision = TP/(TP+FP)
  - Recall = TP/(TP+FN)

	Same cluster	Different clusters
Same class	TP	FN
Different classes	FP	TN

class

ID	Conference Name	Ground Truth Label	Algorithm output Label
1	IJCAI	3	2
2	AAAI	3	2
3	ICDE	1	3
4	VLDB	1	3
5	SIGMOD	1	3
6	SIGIR	4	4
7	ICML	3	2
8	NIPS	3	2
9	CIKM	4	3
10	KDD	2	1
11	WWW	4	4
12	PAKDD	2	1
13	PODS	1	3
14	ICDM	2	1
15	ECML	3	2
16	PKDD	2	1
17	EDBT	1	2
18	SDM	2	1
19	ECIR	4	4
20	WSDM	4	4

we group by ID's and follow table above to get the bellow results

grouping by ID's	TP	TN	FP	FN
	34	143	7	5

(2) Precision $= \frac{34}{34+7} = \frac{34}{41} = \underline{\underline{0.83}}$

(3) Recall $= \frac{34}{34+5} = \frac{34}{39} = \underline{\underline{0.87}}$

(4) F1-score $= \frac{34+143}{34+143+7+5} = \frac{177}{189} = \underline{\underline{0.94}}$

answer 5

5

$\lvert C_i \cap w_j \rvert$	C1	C2	C3	C4	Sum
1	0	0	0	5	5
2	6	0	0	0	6
3	0	5	0	0	5
4	0	0	4	0	4
Sum	6	5	4	5	20

$\lvert w_j \rvert$

$\lvert c_j \rvert$

$$I(C,\Omega) = \frac{5}{20}\log_2\left(\frac{100}{25}\right) + \frac{6}{20}\log_2\left(\frac{120}{36}\right) + \frac{5}{20}\log_2\left(\frac{100}{25}\right) + \frac{4}{20}\log_2\left(\frac{80}{16}\right) = 0.679$$

$$H(C) = -\frac{6}{20}\log_2\frac{6}{20} - \frac{5}{20}\log_2\frac{5}{20} - \frac{5}{20}\log_2\frac{5}{20} - \frac{4}{20}\log_2\frac{4}{20} = 1.98$$

$$H(\Omega) = 4\left(-\frac{1}{4}\log_2\frac{1}{4}\right) = 2$$

$$\sqrt{H(C)H(\Omega)} = 1.98$$

$$NMI = \frac{0.679}{1.98} = \underline{\underline{0.342}}$$

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 [2]:
```python
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 [3]:
```python
k = KMeans()
#=========================#
# STRART YOUR CODE HERE   #
#=========================#
k.main(dataname1)
k.kmeans()
k.main(dataname2)
k.kmeans()
#=========================#
#   END YOUR CODE HERE    #
#=========================#
```

```
For dataset1
Iteration :3
Cluster 0 size :50
Centroid [x=2.5737264423871222, y=-0.027462568841232965]
Cluster 1 size :50
Centroid [x=-0.46333686463472107, y=-0.46611409698195816]
Cluster 2 size :50
Centroid [x=0.988876620573686, y=2.0104789651972013]
Iteration :3
Cluster 0 size :50
Centroid [x=2.5737264423871222, y=-0.027462568841232965]
Cluster 1 size :50
Centroid [x=-0.46333686463472107, y=-0.46611409698195816]
Cluster 2 size :50
Centroid [x=0.988876620573686, y=2.0104789651972013]

For dataset2
Iteration :4
Cluster 0 size :102
Centroid [x=1.2708406269481842, y=-0.08583389704900131]
Cluster 1 size :98
Centroid [x=-0.20185935062367868, y=0.5726963240559536]
Iteration :4
Cluster 0 size :102
Centroid [x=1.2708406269481842, y=-0.08583389704900131]
Cluster 1 size :98
Centroid [x=-0.20185935062367868, y=0.5726963240559536]
```

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 [3]:
```
k = KMeans()
#========================#
# STRART YOUR CODE HERE  #
#========================#
k.main(dataname1)
k.kmeans(True)
k.main(dataname2)
k.kmeans(True)
#========================#
#   END YOUR CODE HERE   #
#========================#
```

```
For dataset1
Iteration :3
Cluster 0 size :50
Centroid [x=2.5737264423871222, y=-0.027462568841232965]
Cluster 1 size :50
Centroid [x=-0.46333686463472107, y=-0.46611409698195816]
Cluster 2 size :50
Centroid [x=0.988876620573686, y=2.0104789651972013]
Iteration :3
Purity is 1.000000
NMI is 1.000000
Cluster 0 size :50
Centroid [x=2.5737264423871222, y=-0.027462568841232965]
Cluster 1 size :50
Centroid [x=-0.46333686463472107, y=-0.46611409698195816]
Cluster 2 size :50
Centroid [x=0.988876620573686, y=2.0104789651972013]

For dataset2
Iteration :4
Cluster 0 size :102
Centroid [x=1.2708406269481842, y=-0.08583389704900131]
Cluster 1 size :98
Centroid [x=-0.20185935062367868, y=0.5726963240559536]
Iteration :4
Purity is 0.760000
NMI is 0.145025
Cluster 0 size :102
Centroid [x=1.2708406269481842, y=-0.08583389704900131]
Cluster 1 size :98
Centroid [x=-0.20185935062367868, y=0.5726963240559536]
```

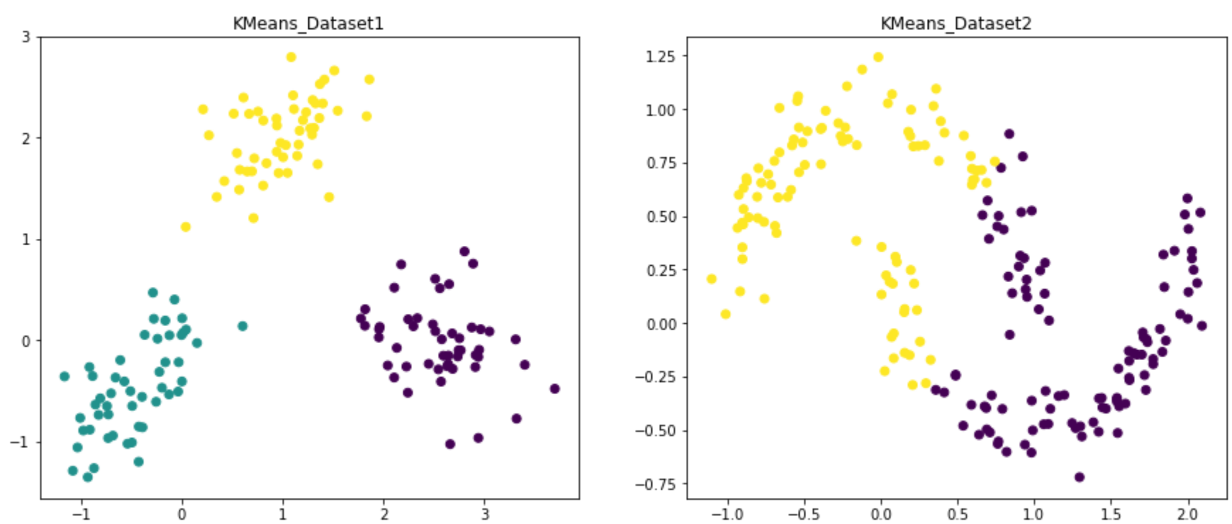
2.3 Visualization

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 [22]:
```
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  #
#========================#
ax0.scatter(df1.iloc[:, 0], df1.iloc[:, 1], c=df1.iloc[:, 2])
ax1.scatter(df2.iloc[:, 0], df2.iloc[:, 1], c=df2.iloc[:, 2])
#========================#
#   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: 1. It's efficient as it has a linear run time

2. It's easy to interpret

Cons: Not suitable to discover clusters with non-convex shapes. 2 It's sensitive to noisy data

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 [10]:
```
from hw4code.DBSCAN import DBSCAN
d = DBSCAN()
#========================#
# STRART YOUR CODE HERE  #
#========================#
d.main(dataname1)
d.main(dataname2)
#========================#
#   END YOUR CODE HERE   #
#========================#
```

```
For dataset1
Esp :0.3560832705047313
Number of clusters formed :4
Noise points :9
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
```

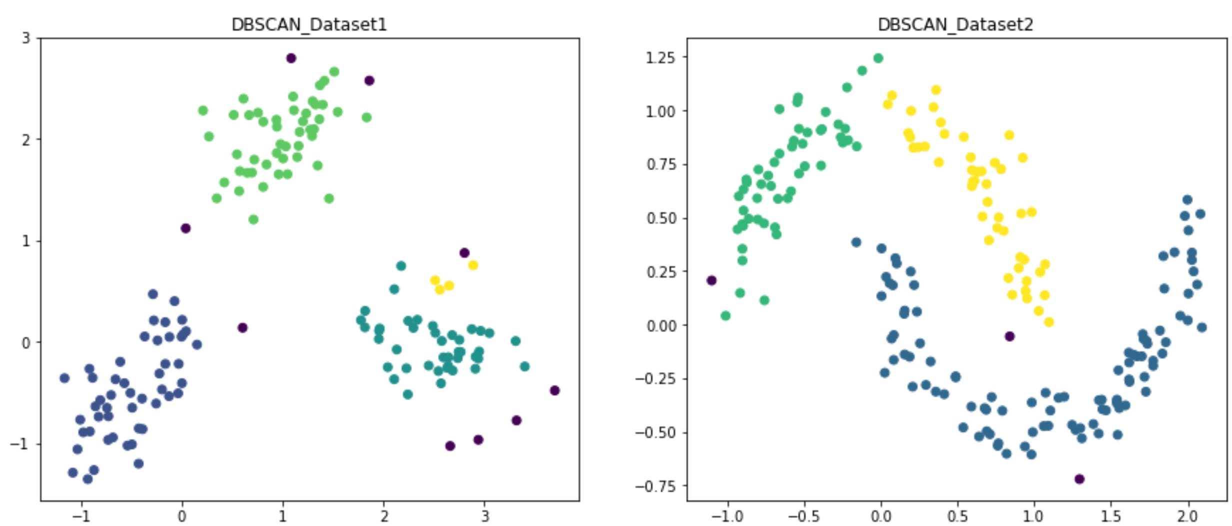
3.2 Visualization

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 [11]:
```
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  #
#========================#
ax0.scatter(df1.iloc[:, 0], df1.iloc[:, 1], c=df1.iloc[:, 2])
ax1.scatter(df2.iloc[:, 0], df2.iloc[:, 1], c=df2.iloc[:, 2])
#========================#
#   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: It can find clust with aribitarely shapes. 2 It's robust to outliers

Cons: Its outputs relies on its parameters and it's difficult to find the optimal parameters

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 [12]:
```
from hw4code.GMM import GMM
g = GMM()
#========================#
# STRART YOUR CODE HERE  #
#========================#
g.main(dataname1)
g.main(dataname2)
#========================#
#   END YOUR CODE HERE   #
#========================#
```

```
For Cluster : 1
0.7692790765358335
-0.28782809642382123

-0.28782809642382123
0.1901249384356512


For Cluster : 2
0.6828574757628689
-0.30058915994390517

-0.30058915994390517
0.17583559485120062


Purity is 0.690000
NMI is 0.075948
Cluster 0 size :106
Cluster 1 size :94
```

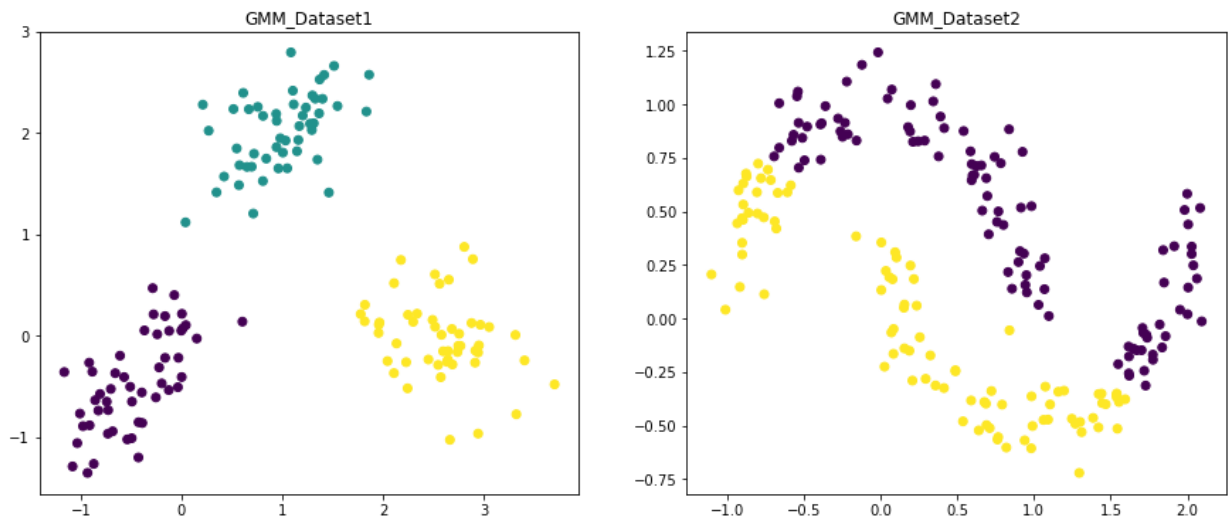
4.2 Visualization

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 [13]:
```
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  #
#========================#
ax0.scatter(df1.iloc[:, 0], df1.iloc[:, 1], c=df1.iloc[:, 2])
ax1.scatter(df2.iloc[:, 0], df2.iloc[:, 1], c=df2.iloc[:, 2])
#========================#
#   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 models are more general than partitioning: different densities and sizes of clusters. 2. Clusters can be characterized by a small number of parameters

Cons of GMM: 1. Converge to local optimal. 2. Hard to estimate the number of clusters

Reasoning over dataset1: Kmeans and GMM resulted in 1,0 purity whereras DBSCAN in 0.94. And we can perfectly see in the visualization as there are three perfect clusters. However, for DBSCAN we can see there are some points identified as Noise and there is a fourth cluster as well. Maybe by tunning DBSCAN parameters we can get a better purity and cluster creation for DBSCAN

Reasoning over dataset2: DBSCAN gets the best purity, 0.98, then Kmeans with 0.88 and lastly GMM with 0.69 as the lowest one. I think DBSCAN performed the best because clusters are dense and seperabalem Keamns is still perform relatively good and the reason it's not as good. Lastly we can see why GMM is not performing well due to its constraint on performing on non-convex shapes

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

```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)]
```

```python
    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]
```

```
            # ========================#
            #   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  #
        # ========================#
            HOmega += nmiMatrix[row - 1][j]/float(N) * math.log(nmiMatrix[row - 1][j] / float(N))
        # ========================#
        #   END YOUR CODE HERE   #
        # ========================#

    #Compute HC
    # ========================#
    # STRART YOUR CODE HERE  #
    # ========================#
        HC += nmiMatrix[i][col - 1]/float(N) * math.log(nmiMatrix[i][col - 1]/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
    # ---------------------------------------------------------------------
```

```
        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]))

```

```python

        # 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   #
            # ========================#
            for cluster in clusters:
                if point in cluster:
                    cluster.remove(point)
```

```python
            # ========================#
            #   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  #
        # ========================#
        for data_point in cluster:
            # print(data_point)
            cx += data_point.x
            cy += data_point.y
        cx /= len(cluster)
        cy /= 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
```