Amirkabir University of Technology
(Tehran Polytechnic)

Computer Engineering Department

Machine Learning Course
CE5501

Homework 5

Student
Reza Sajedi
400131072
r.sajedi@aut.ac.ir

Teacher
Dr. Nazerfard

Fall 2022

Table of Contents

Problem 1: Why and how

Part a

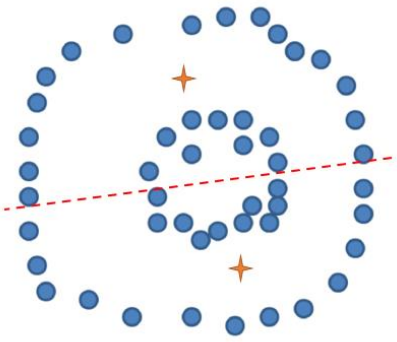
Since the clustering algorithms are dealing with data points distances, generally we can set a threshold for the distance and consider far data points from the nearest cluster as outliers. For example, in K-means after calculating the distance of each data point from its nearest centroid, we can consider the top N farthest data points as outliers. In DBSCAN, the data points that are not in the epsilon distance neighborhood of any cluster are considered outliers. In GMM, each data point is assigned to a cluster with a probability. We can simply consider data points with low probabilities as outliers.

Part b

EM works the same way as K-means except that the data is assigned to each cluster with the weights being soft probabilities instead of distances. The advantage is that the model becomes generative as we define the probability distribution for each model. The EM algorithm alternates between two steps (E-step and M-step). In the E-step the algorithm tries to find a lower bound function on the original likelihood using the current estimate of the statistical parameters. In the M-step the algorithm finds new estimates of those statistical parameters by maximizing the lower bound function (i.e., determining the MLE of the statistical parameters). Since at each step, we maximize the lower bound function, the algorithm always produces estimates with a higher likelihood than the previous iteration and ultimately converges to a maximum. Unlike K-means, the clusters are not limited to spherical shapes in EM. In EM we can constrain the algorithm to provide different covariance matrices (spherical, diagonal, and generic). These different covariance matrices in return allow us to control the shape of our clusters; hence, we can detect sub-populations in our data with different characteristics. We can say that K-Means is the specific form of EM, which consist of K Gaussian functions.

Part c

I think for this data set, the centers of the two clusters found with two algorithms are the same. None of them can be used to accurately separate concave data sets. There are suitable for linearly separable data sets.



Part d

Some studies have compared the results of the various clustering algorithms for market segmentation. The empirical results show that K-means has the highest performance in this application. So, it seems that the density-based algorithms are the better choice because in most cases the clusters are linearly-separable. Although I think it's better to test various algorithms for each dataset and choose the one with the highest performance according to the chosen evaluation metric.

Part e

ID	*{1}*	*{2}*	*{3}*	*{4}*	*{5}*
X	14	18	25	34	34
Y	6	6	10	6	14

Calculating the distance matrices using the Euclidean metric:

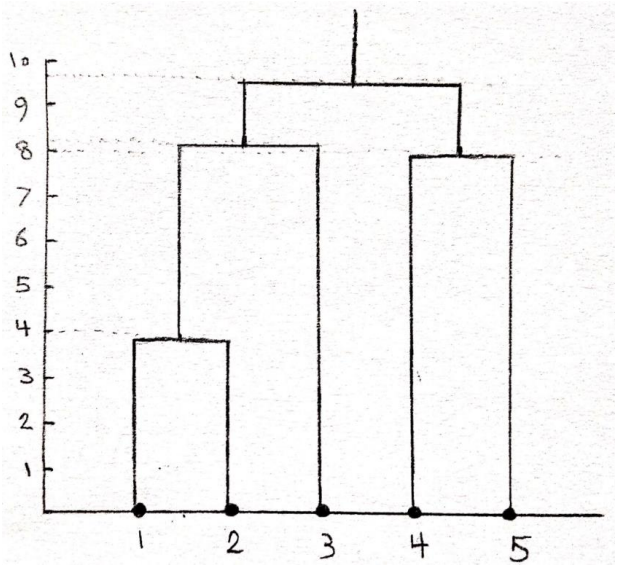
	{1}	{2}	{3}	{4}	{5}
{1}	0				
{2}	4	0			
{3}	11.7	8.0	0		
{4}	20	16	9.8	0	
{5}	21.5	17.8	9.8	8	0

	{1,2}	{3}	{4}	{5}
{1,2}	0			
{3}	8.0	0		
{4}	16	9.8	0	
{5}	17.8	9.8	8	0

	{1,2}	{3}	{4,5}
{1,2}	0		
{3}	8.0	0	
{4,5}	16	9.8	0

	{1,2,3}	{4,5}
{1,2,3}	0	
{4,5}	9.8	0

	{1,2,3,4,5}
{1,2,3,4,5}	0



Part f

In this situation, all data points will be assigned to one cluster and the other cluster will be empty. In the next iterations that cluster may remain empty. So, convergence will be done with only one cluster. We can prevent this problem by choosing the initial centers from the data points and even for better results selecting them as far as possible.

```
Init:
    center1 = -5
    center2 = 1

Iter1:
    |-12 + 5| = 7,     |-12 - 1| = 13
    |-11 + 5| = 6,     |-11 - 1| = 12
    |-9 + 5| = 4,      |-9 - 1| = 10
    |-7 + 5| = 2,      |-7 - 1| = 8
    |-6 + 5| = 1,      |-6 - 1| = 7
    |4.5 + 5| = 9.5,   |4.5 - 1| = 3.5
    |7.5 + 5| = 12.5,  |7.5 - 1| = 6.5
    |10 + 5| = 15,     |10 - 1| = 9

    c1 = {-12, -11, -9, -7, -6}
```

```
    c2 = {4.5, 7.5, 10}

    center1 = (-12 - 11 - 9 - 7 - 6) / 5 = -9
    center2 = (4.5 + 7.5 + 10) / 3 = 7.33

Iter2:
    |-12 + 9| = 3,    |-12 - 7.33| = 19.33
    |-11 + 9| = 2,    |-11 - 7.33| = 18.33
    |-9 + 9| = 0,     |-9 - 7.33| = 16.33
    |-7 + 9| = 2,     |-7 - 7.33| = 14.33
    |-6 + 9| = 3,     |-6 - 7.33| = 13.33
    |4.5 + 9| = 13.5, |4.5 - 7.33| = 2.83
    |7.5 + 9| = 16.5, |7.5 - 7.33| = 0.17
    |10 + 9| = 19,    |10 - 7.33| = 2.67

    c1 = {-12, -11, -9, -7, -6}
    c2 = {4.5, 7.5, 10}

    center1 = (-12 - 11 - 9 - 7 - 6) / 5 = -9
    center2 = (4.5 + 7.5 + 10) / 3 = 7.33

    center1 = previous_center1 & center2 = previous_center2 ---> Converged!
```

Part g

GMM is a probabilistic model that assumes all the data points are generated from a mix of Gaussian distributions with unknown parameters. They can be used to find clusters in data sets where the clusters may not be clearly defined. The goal of the algorithm is to estimate the parameters of the Gaussian distributions, as well as the proportion of data points that come from each distribution. They can analyze more complex and mixed data and they are relatively robust to outliers. It is difficult to directly interpret results and they don't directly assign data points to clusters. GMM: uses three parameters: the number of clusters K, mean, and cluster covariances.

Hyperparameters:

- Number of clusters: The number of clusters to group data into.

- Maximum iterations: a fixed number of cycles for the algorithm (this may end the algorithm sooner than a convergence)

- Number initial (n_init): number of times the whole algorithm is run, with different starting locations for the centroids. Since the algorithm is sensitive to starting points, it allows you to select several different starting points to seek the optimal solution.

Part h

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha.\left(r_t + \gamma.max_a Q(S_{t+1}, a) - Q(s_t, a_t)\right)$$

$$Q_{new}\big((3,2), Down\big) = 0 + 0.1 \times (-0.3 + 0.9 \times 0.8 - 0)$$

$$Q_{new}\big((3,2), Down\big) = 0.042$$

Problem 2: Blood Analysis | HCV

Part a

If we don't normalize, some features with large-scale values may conquer other features and neutralize their effects. Also, numerical features may have different units. For example, if we want to cluster people on their weights in kilograms and heights in meters, a 1kg difference is not as significant as a 1m difference in height. So, we need to normalize the dataset before applying distance-based clustering algorithms to assign equal weights to the features.

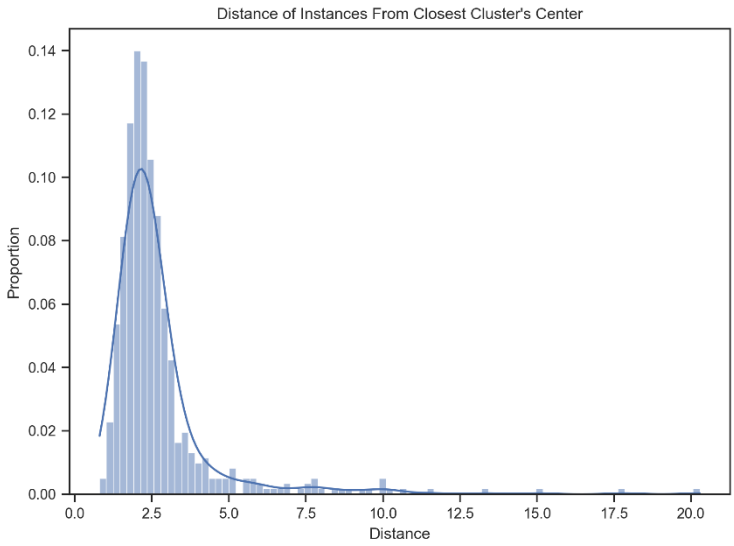
Part b

The instruction has been followed.

Part c

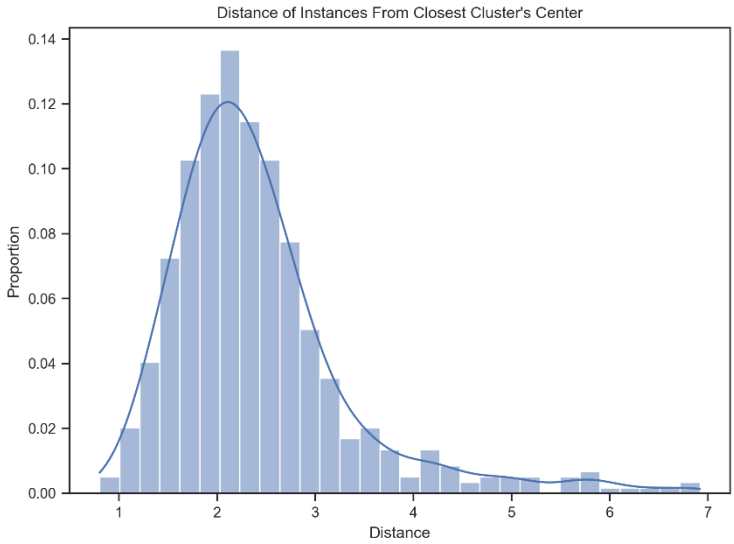
Data points which have high distances from the clusters may be considered outliers. We can check the distribution of the distances using a Histogram diagram to detect outliers.

```
Dataset status before removing outliers:
Number of instances in each cluster (y_pred): {0: 364, 1: 251}
Number of instances in each class (y_true): {0: 540, 1: 24, 2: 21, 3: 30}
```



According to the above diagram, data points which have distances higher than 7.0, may be abnormal because they differ significantly from the major points. After removing such points which constitute 3.5% of the data set, we get the following result:

```
Dataset status after removing outliers:
Number of instances in each cluster (y_pred): {0: 361, 1: 232}
Number of instances in each class (y_true): {0: 536, 1: 22, 2: 20, 3: 15}
The proportion of outliers: 0.0358
```



Part d

The instruction has been followed.

Part e

```
Results for Euclidean:
Purity: 0.9076
Entropy: 0.4445
Accuracy: 0.5546
Centers:
[[ 0.0919 -0.923  -0.4066 -0.1159 -0.3178 -0.2336 -0.125  -0.4203 -0.0891
  -0.2203 -0.2782 -0.2939]
 [-0.1091  0.7019  0.4509 -0.0143  0.1852 -0.0198 -0.0503  0.402   0.1467
   0.0614  0.0099  0.2907]]


Results for CityBlock:
Purity: 0.9076
Entropy: 0.4394
Accuracy: 0.521
Centers:
[[-0.2049  0.6618  0.4981 -0.0261  0.2626  0.0428 -0.029   0.494   0.1584
   0.0696  0.0619  0.4267]
 [ 0.1556 -0.6562 -0.3314 -0.0904 -0.3213 -0.2636 -0.1349 -0.3931 -0.0672
  -0.1886 -0.2875 -0.3421]]
```

Part f

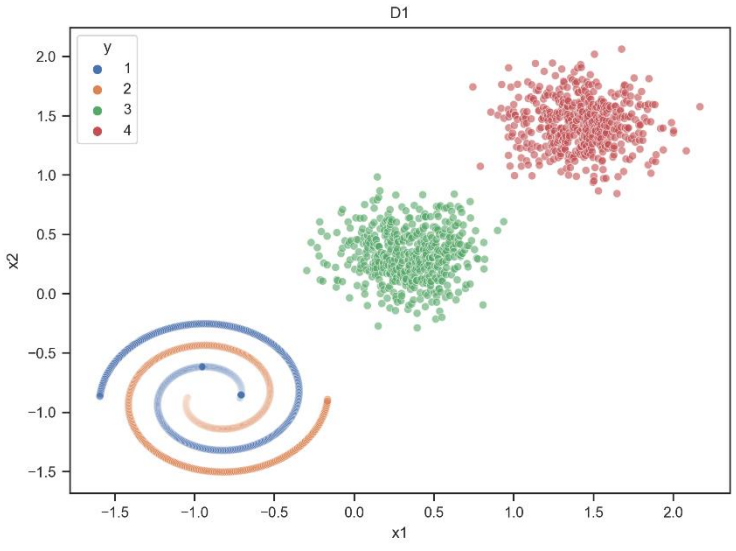
```
Results for Euclidean:
Purity: 0.9076
Entropy: 0.5128
Accuracy: 0.3529
Centers:
[[-4.4094e-02 -1.2586e+00 -7.2146e-02 -1.1434e-01 -3.2487e-01 -3.3597e-01
  -2.2311e-01 -1.9316e-01  1.3054e-01 -2.2433e-01 -3.1323e-01  3.3222e-04]
```

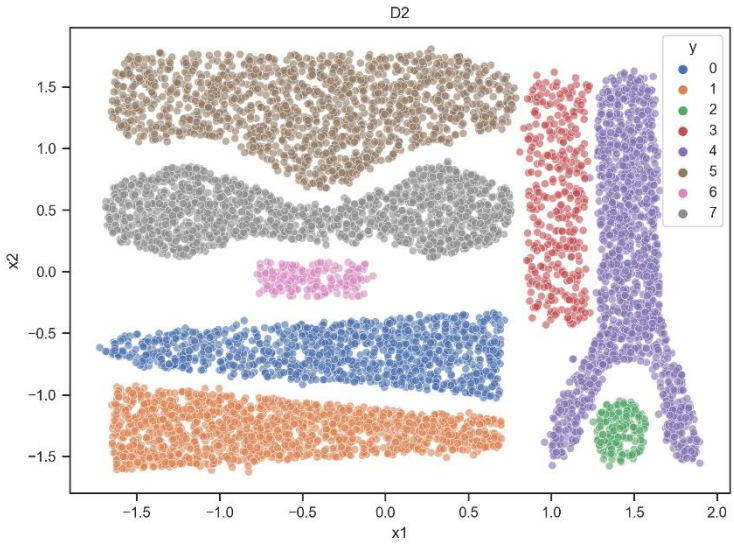
```
 [ 2.4205e-01  7.3295e-01  3.0070e-01  2.6591e-01  5.8262e-01  3.9429e-02
  -7.0801e-02  8.1864e-01  8.8700e-01  1.1703e-01  2.7220e-01  4.7958e-01]
 [-4.6167e-01  7.9454e-01  4.7569e-01 -1.9734e-01 -8.5691e-02 -5.7668e-02
  -4.1034e-02  1.7461e-01 -3.6383e-01  2.3536e-02 -1.4836e-01  1.6439e-01]
 [ 8.7309e-01  3.4286e-01 -1.0918e+00 -9.6856e-02 -1.2397e-01  1.7492e-01
   2.4608e-01 -1.1139e+00 -7.0732e-01 -1.2378e-01 -1.2042e-01 -1.1360e+00]]


Results for CityBlock:
Purity: 0.9244
Entropy: 0.384
Accuracy: 0.3866
Centers:
[[-0.1567  0.7225  0.6246  0.0674  0.2897  0.0329 -0.0359  0.7792  0.7494
   0.1099  0.179   0.5976]
 [-0.2455  0.4322  0.0484 -0.248   2.7481  1.4877  0.0328 -0.3188 -0.6512
  -0.1564  0.7943  0.3351]
 [ 0.0044 -1.2586 -0.1986 -0.0794 -0.3405 -0.3155 -0.2093 -0.2569  0.0862
  -0.244  -0.2984 -0.0901]
 [ 0.0692  0.7945 -0.0068 -0.1047 -0.2173 -0.1612  0.0099 -0.1094 -0.4226
   0.0263 -0.2178 -0.2529]]
```

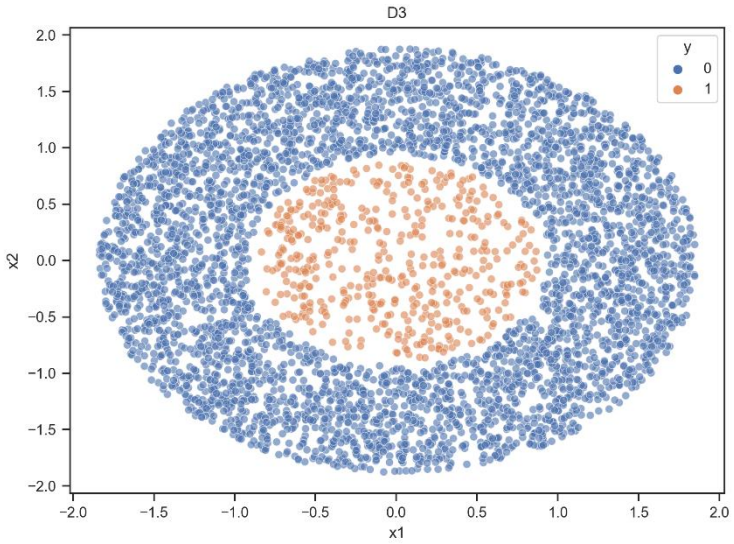
Problem 3: DBScan is talking

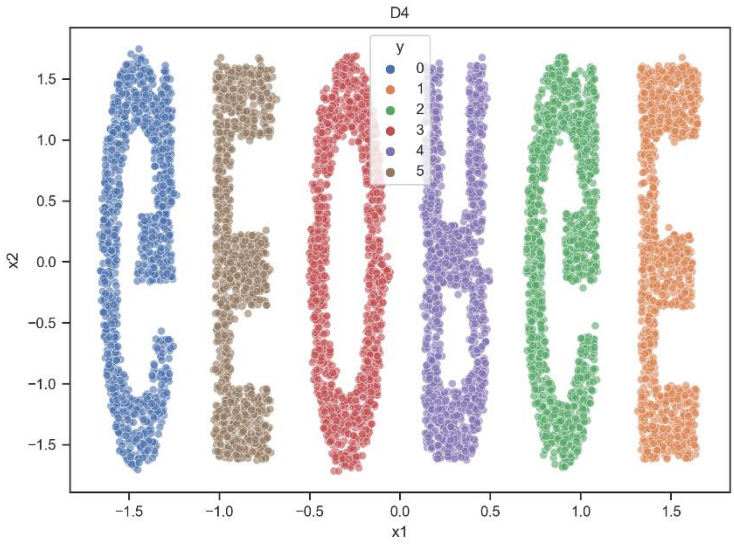
Part a

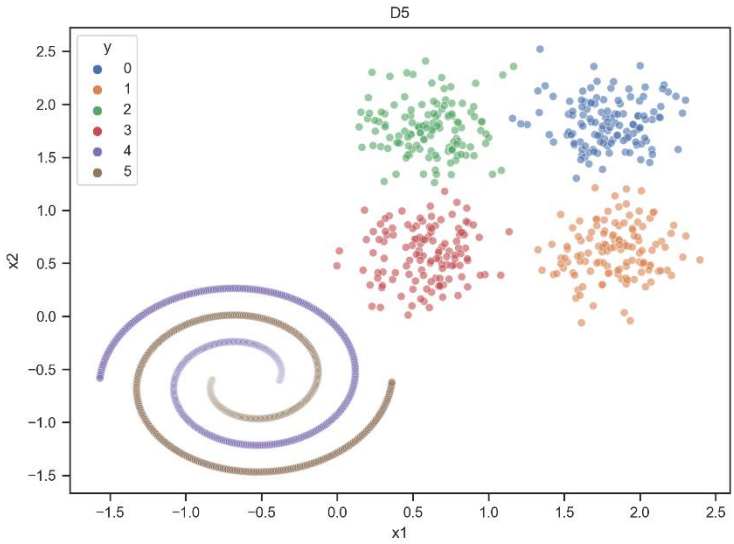
Different clusters for each data set have been shown with different unique colors in each figure. So, the number of clusters in each cluster is obvious.




D2
y
0
1
2
3
4
5
6
7
x2
x1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


D3
y
0
1
x2
x1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


D4
y
0
1
2
3
4
5
x2
x1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

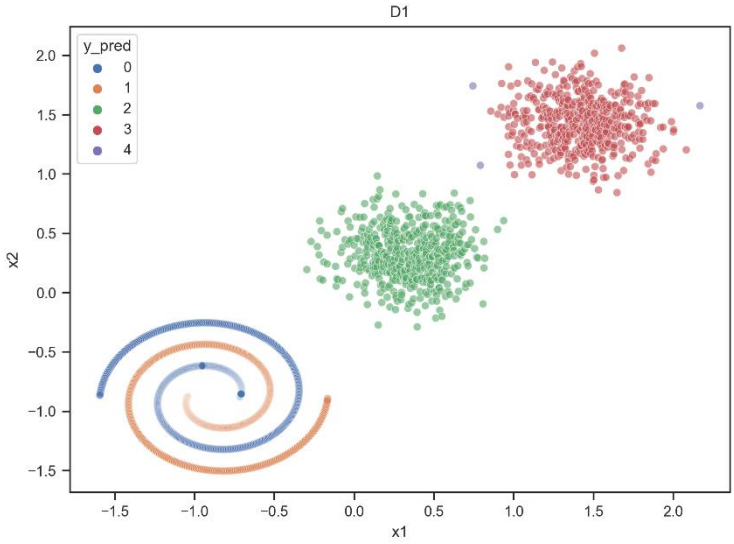


Part b

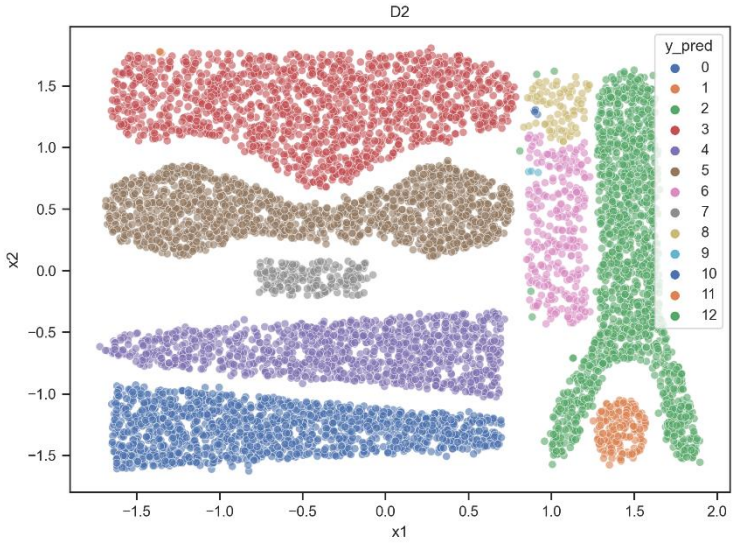
The instruction has been followed.

Part c

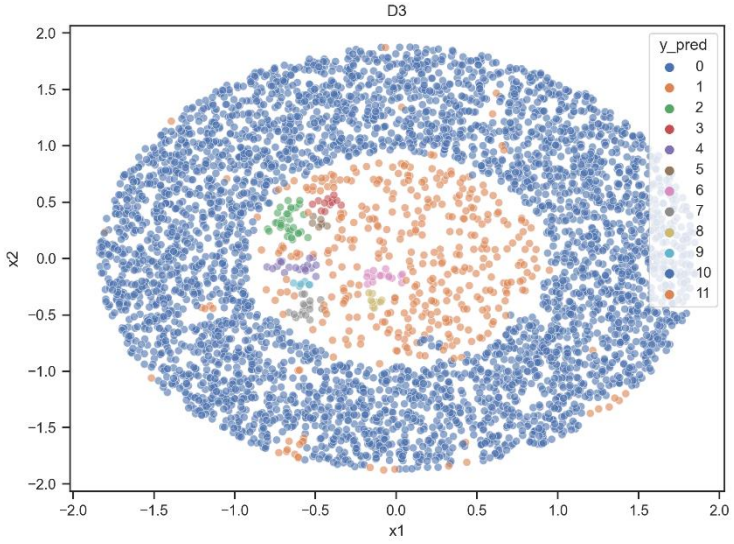
```
Results for D1:
Epsilon: 0.177, MinPoints: 3
Number of instances in each cluster (y_pred):
{0: 500, 1: 500, 2: 500, 3: 497, 4: 3}
```



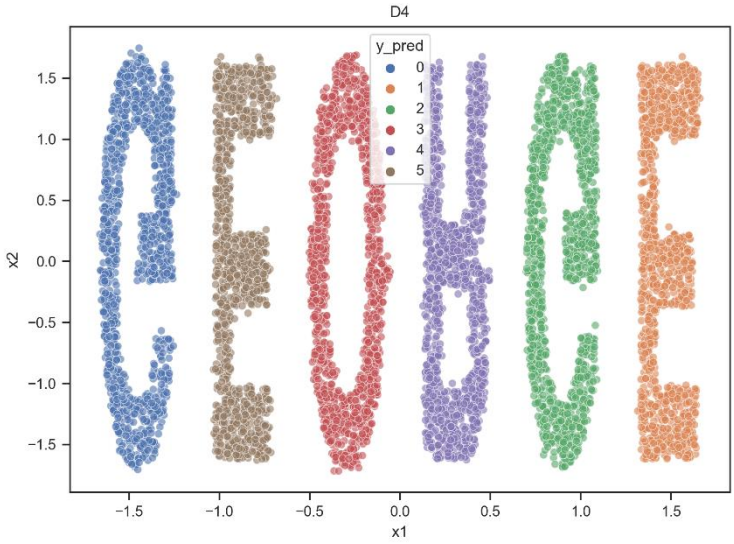
Results for D2:
Epsilon: 0.0643, MinPoints: 3
Number of instances in each cluster (y_pred):
{0: 1455, 1: 182, 2: 1389, 3: 1449, 4: 1112, 5: 1557, 6: 263, 7: 175, 8: 77, 9: 3, 10: 4, 11: 3, 12: 8}



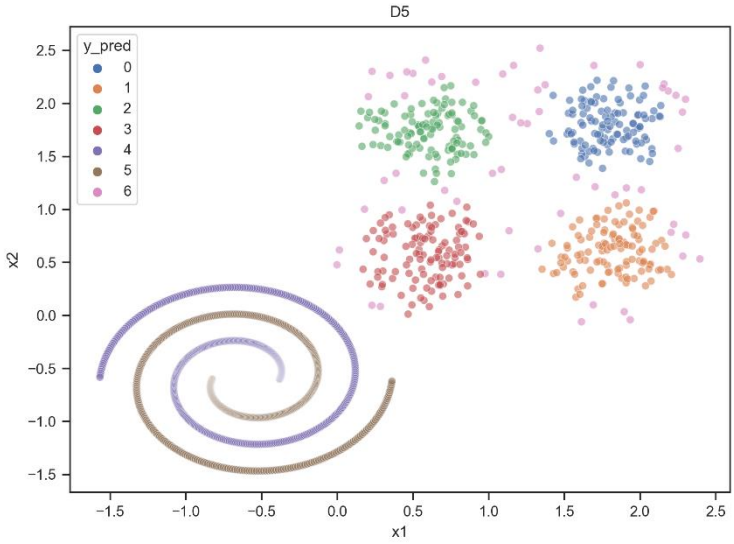
Results for D3:
Epsilon: 0.08, MinPoints: 8
Number of instances in each cluster (y_pred):
{0: 4091, 1: 10, 2: 32, 3: 12, 4: 19, 5: 7, 6: 15, 7: 16, 8: 9, 9: 6, 10: 8, 11: 375}



```
Results for D4:
Epsilon: 0.1, MinPoints: 5
Number of instances in each cluster (y_pred):
{0: 1173, 1: 1053, 2: 1198, 3: 1163, 4: 1206, 5: 1054}
```




```
Results for D5:
Epsilon: 0.13, MinPoints: 6
Number of instances in each cluster (y_pred):
{0: 108, 1: 111, 2: 110, 3: 112, 4: 500, 5: 500, 6: 59}
```

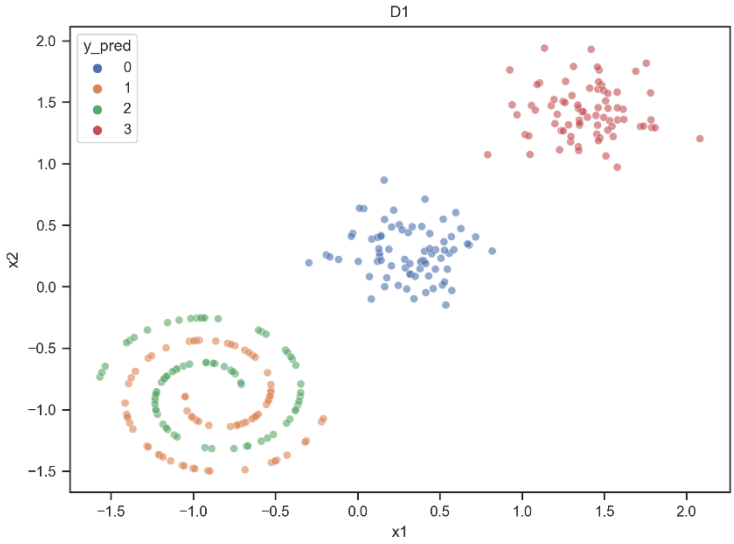


Part d

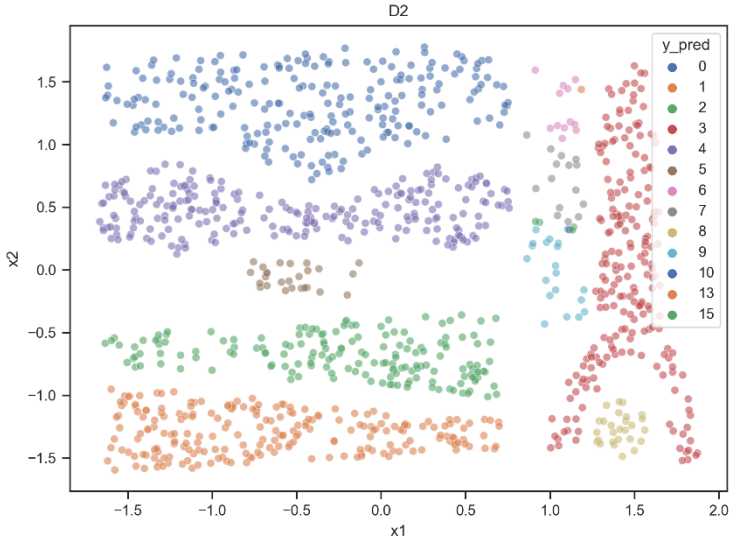
Our strategy for assigning a cluster to a sample data point is that we find the nearest data point from the train data set and consider its cluster for our sample.

Part e

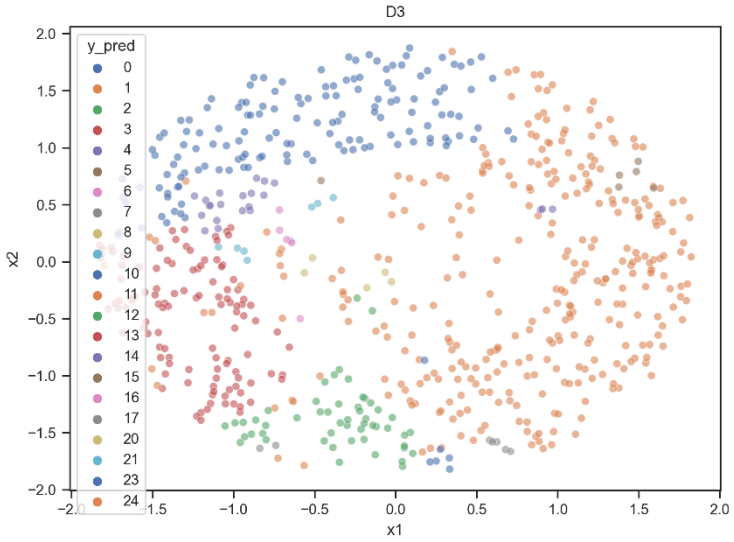
```
Results for D1 with 1-NN:
Epsilon: 0.177, MinPoints: 3
Purity: 1.0
Entropy: 0.0
Accuracy: 0.75
```



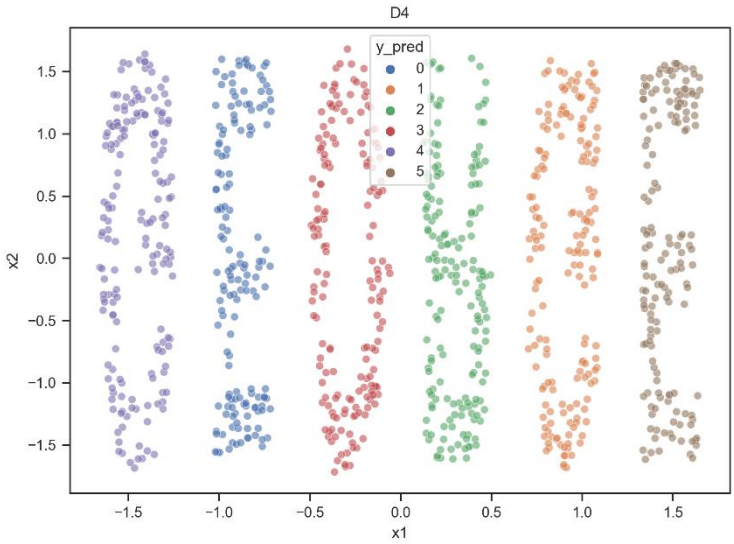

```
Results for D2 with 1-NN:
Epsilon: 0.0643, MinPoints: 3
Purity: 1.0
Entropy: 0.0
Accuracy: 0.9705
```



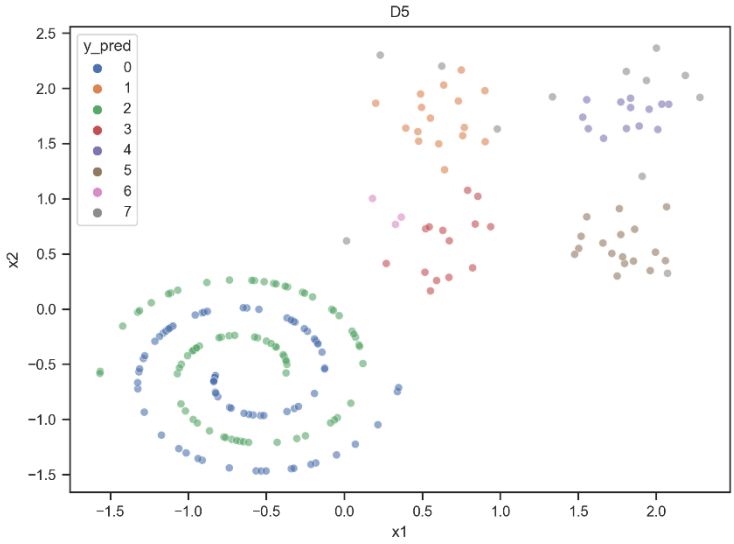
Results for D3 with 1-NN:
Epsilon: 0.08, MinPoints: 8
Purity: 0.9464
Entropy: 0.1252
Accuracy: 0.4072



Results for D4 with 1-NN:
Epsilon: 0.1, MinPoints: 5
Purity: 1.0
Entropy: 0.0
Accuracy: 1.0



```
Results for D5 with 1-NN:
Epsilon: 0.13, MinPoints: 6
Purity: 0.9733
Entropy: 0.0922
Accuracy: 0.9333
```



Part f

The results show that using 3-NN and 5-NN has minor improvements for some data sets compared to the previous part. So, this way is preferred. Because by choosing a suitable value for K, the prediction becomes more robust against outliers. If we only consider a single nearest neighbor, this point may be an outlier and we will assign a wrong cluster to our sample data point.

```
Results for D1 with 3-NN:
Epsilon: 0.177, MinPoints: 3
Purity: 1.0
Entropy: 0.0
Accuracy: 0.75


Results for D2 with 3-NN:
Epsilon: 0.0643, MinPoints: 3
Purity: 1.0
Entropy: 0.0
Accuracy: 0.9714


Results for D3 with 3-NN:
Epsilon: 0.08, MinPoints: 8
Purity: 0.9565
Entropy: 0.1319
Accuracy: 0.4087


Results for D4 with 3-NN:
```

```
Epsilon: 0.1, MinPoints: 5
Purity: 1.0
Entropy: 0.0
Accuracy: 1.0


Results for D5 with 3-NN:
Epsilon: 0.13, MinPoints: 6
Purity: 0.9867
Entropy: 0.0462
Accuracy: 0.9467



Results for D1 with 5-NN:
Epsilon: 0.177, MinPoints: 3
Purity: 1.0
Entropy: 0.0
Accuracy: 0.75


Results for D2 with 5-NN:
Epsilon: 0.0643, MinPoints: 3
Purity: 1.0
Entropy: 0.0
Accuracy: 0.9731


Results for D3 with 5-NN:
Epsilon: 0.08, MinPoints: 8
Purity: 0.971
Entropy: 0.0948
Accuracy: 0.4087


Results for D4 with 5-NN:
Epsilon: 0.1, MinPoints: 5
Purity: 1.0
Entropy: 0.0
Accuracy: 1.0


Results for D5 with 5-NN:
Epsilon: 0.13, MinPoints: 6
Purity: 0.9867
Entropy: 0.0462
Accuracy: 0.9467
```

Problem 4: Frozen Lake

Part a

The environment is composed of tiles, where the agent has to reach the goal from the start tile. Tiles can be a safely frozen lake or a hole that gets the agent stuck forever. The agent can perform 4 possible actions: go LEFT, DOWN, RIGHT, or UP. The agent must learn to avoid holes to reach the goal in a minimal number of actions. In reinforcement learning, agents are rewarded by the environment when they accomplish a predefined goal. In Frozen Lake, the agent is only rewarded when it reaches the state G. The reward is 1 when the agent reaches G, and 0 otherwise. There are two versions of the game: one with slippery ice, where selected actions have a random chance of being disregarded by the agent; and a non-slippery one, where actions cannot be ignored.

Part b

```
Number of actions: 4
Number of observations: 16

Win rate: 0.009
Number of wins: 9
```

Part c

The instruction has been followed.

Part d

We have 4 actions and 16 states (tiles). So, we generate a 16 x 4 table called Q-table which all of its cells will be initialized with zero. This table is trained during several episodes. When the agent moves by an action to the next state, the value corresponding to the previous state and action is updated in the table using the following equation:

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha.\left(r_t + \gamma.max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$$

Where α is the learning rate which balances between the importance of past and new knowledge and γ is the discount factor which determines how much the agent cares about future rewards compared to immediate ones.

In each state, the agent uses the Epsilon-Greedy algorithm to take the next action. The action with a higher value is taken with a higher probability. This method let us both exploit the best current obtained ways and explore even better ones.

```
Q-table:
[[0.5303 0.5905 0.5453 0.5295]
 [0.3349 0.     0.6333 0.2903]
 [0.2797 0.7225 0.141  0.3965]
 [0.3313 0.     0.0453 0.0395]
 [0.5885 0.6561 0.     0.5298]
 [0.     0.     0.     0.    ]
 [0.     0.8096 0.     0.3792]
 [0.     0.     0.     0.    ]
 [0.6513 0.     0.729  0.5857]
 [0.6437 0.7935 0.81   0.    ]
 [0.7153 0.9    0.     0.7094]
 [0.     0.     0.     0.    ]
 [0.     0.     0.     0.    ]
 [0.     0.5566 0.8995 0.446 ]
 [0.7902 0.891  1.     0.7967]
 [0.     0.     0.     0.    ]]
```

Part e

The agent won all of the episodes using this policy. The random policy used in part b had a very low win rate.

```
Win rate: 1.0
Number of wins: 1000
```

Part f

The animation file (*P4_f.gif*) has been attached.