CSCI-550 HW2 - Report

Team Members:

Adiesha Liyanage, Kaveen Liyanage, Siddat Nesar

A. HOW TO RUN THE SYSTEM

First we need to create the synthetic data. It can be generated by running synthetic_data.py. Which we already ran and created the synthetic data. We have created 9 different synthetic data sets to run.

Command: python synthetic_data.py

The above command will generate a dataset with 5 clusters, with a total of 700 sample points, of which 179 points are noises.

Line 77 of the script is responsible for the number of polygons, x-axis range, y-axis range, and number of sample points. An example is given below.

```
abc = DataWithLabel(5, [0, 500], [0, 500], 700)
```

To change the number of noise points as desired, line 40 needs to be modified. The integer value is responsible for the manipulation of the noise points. An example is given below.

```
estimate_points = int(self.num_points / (self.num_poly + 3))
```

To print the number of noises, line be 83 to 88 can be commented out.

We run DBSCAN algorithm experiments by executing the dbscanAssessment.py script. We had included different parameters to run inside this script. It will automatically run the iris data and synthetic data and output the purity and silhouette coefficients values. When we run this script it will automatically generate the plot of Silhouette coefficients against sorted cluster points.

Command: python dbscanAssessment.py

K_means.py has three methods

The *data* should be a pandas dataframe with initial columns, the attributes and the last column, the *label* information.

K_means.k_means(data, K, epsilon) Calculates the K-means clustering for the given *K* clusters until the sum of squared error of the centroid change is less than the given *epsilon* value. Centroid initialization is done according to k-means++ algorithm. This will return

K_means.K_means_w_epochs(data, K, epsilon, epochs) Calculates multiple epochs of K-means algorithm according to **K_means.k_means(df, K, epsilon)** with the given *epochs* number. The best epoch is chosen by the epoch which has the highest *silhouette coefficient*.

K_means.K_means_find_parameters(data, kRange, epsRange, epochs) Calculates **K_means_w_epochs(data, K, epsilon, epochs)** for all the combination of values in *K* values in *kRange* and *epsilon* values in *epsRange* . The best *K* and *epsilon* parameters are determined by the highest *silhouette coefficient* value. (e.g. kRange = [2, 3, 4, 5, 6, 7, 8, 9, 10], epsRange = [0.1, 0.01], epochs = 3)

k_meansAssessment.py will apply the K-means algorithm to Iris and synthetic datasets and plots the resulting silhouette coefficients.

B. EXAMPLE INPUT AND OUTPUT

Command: python synthetic_data.py

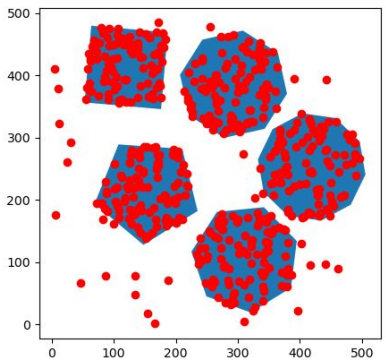
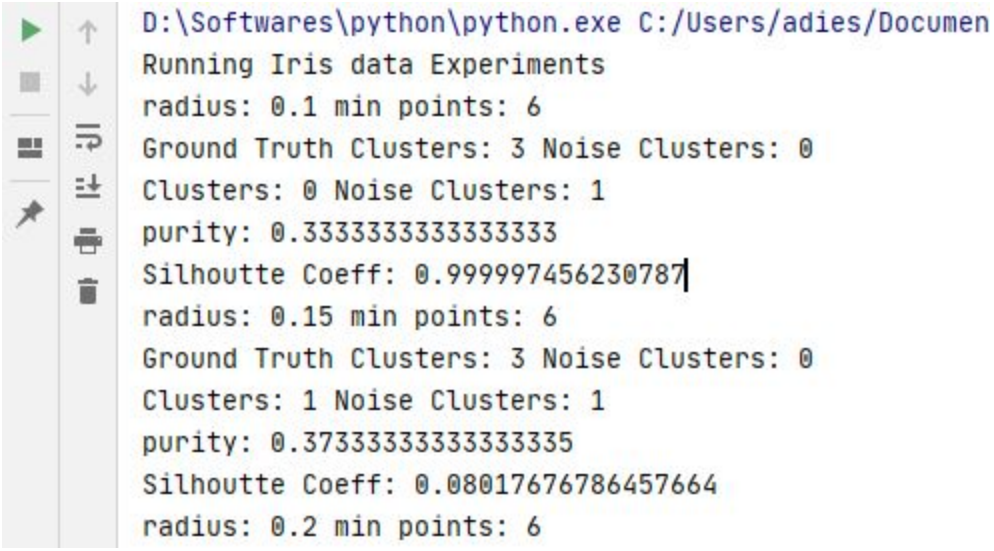
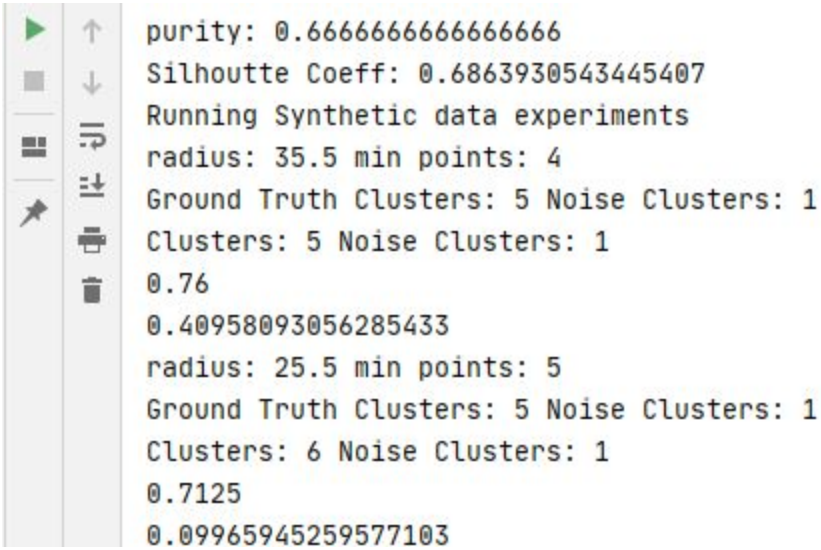


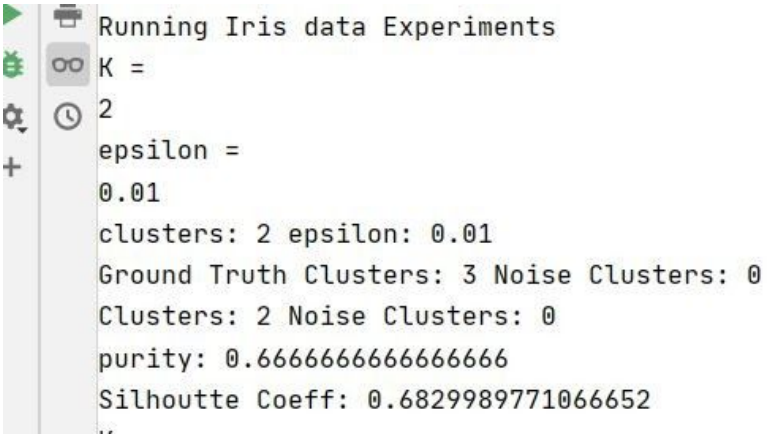
Fig. 1.1 Example plot of synthetic data

Command: python dbscanAssessment.py





Command: python k_meansAssessment.py



C. SSH URI OF THE GIT REPO

The ssh uri for the project is: git@github.com:adiesha/UnsupervisedLearning.git

D. DISCUSSION ABOUT THE PROJECT

In this project, we have implemented two unsupervised clustering methods. First, is a representation based **k-means** clustering technique. Second is a density-based **DBScan** clustering technique. Then two clustering assessment metrics were implemented, first, an internal metric called the **Silhouette coefficient** which doesn't use ground-truth label data, and second, an external metric called **Purity** which uses ground truth label information for cluster assessment. To test the clustering techniques we have used two datasets, **Iris dataset,** and a **Synthetic dataset.** All the implementation is done in a python environment.

This implementation allows someone to compare the two clustering methods for any given datasets with the two assessment metrics. Which allows getting an understanding of data distribution.

- **The Iris Datasets**

The Iris dataset is provided by UC Irvine Machine Learning Repository. The dataset is used to classify three classes of the Iris plant. The dataset consists of 150 samples with 4 attributes.

- **The synthetic dataset**

Synthetic data are artificially generated data which are not obtained from real - world events. The data is created keeping the basic properties similar to real-world data.

This type of data helps to have more data to test the accuracy and efficiency of the algorithm. The focus of this problem is to classify the clustered data, so the synthetic data has been generated in some polygon shaped clusters, along with some noises. The following gives a brief about the algorithm used to generate the synthetic data:

The designed algorithm (**synthetic_data.py**) can take parameters like, number of polygons, range of the boundary region, and number of sample points.

First, the centers of the polygon need to be generated, and thus another boundary has been generated which is far from the given boundary by a distance equal to the radius of the polygons. The **opt_center_points.py** script generates random points within the inner boundary region which shall be considered as the centers of the polygons. To get the optimal maximum distance between the centers, the algorithm iterates 10000 times. Considering those points as the centers, (**generate_polygon.py**) generates the polygons with random shapes. The radius of the polygons has been kept the same to get the biggest clusters possible. The **synthetic_data.py** script checks the points lying within the polygon regions and assigns them labels. The remaining points are considered as noise points. With minor changes in this script, the number of noise points can be manipulated as desired. Several synthetic data were generated to verify the authenticity of the algorithm. The generated synthetic data have the name format as: **Synthetic_NumberofSamplePoints_NumberofNoisePoints**. Those data also contain the respective labels of the data points. The noise points were labeled as 0. For this dataset, 5 polygons/clusters were taken into consideration. Illustrations of the synthetic dataset are furnished below:

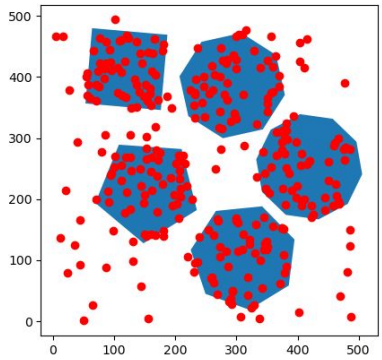


Fig. 2.1. 300 sample points with 63 noise points

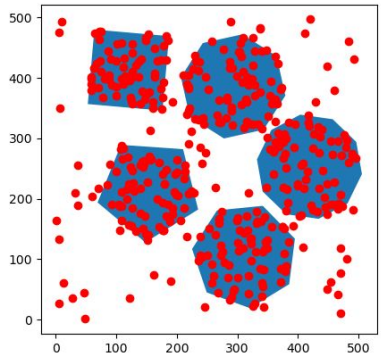


Fig. 2.2. 400 sample points with 63 noise points

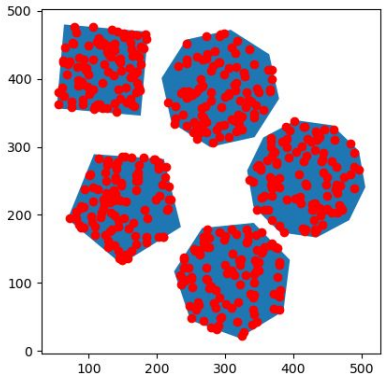


Fig. 2.3. 500 sample points with 0 noise points

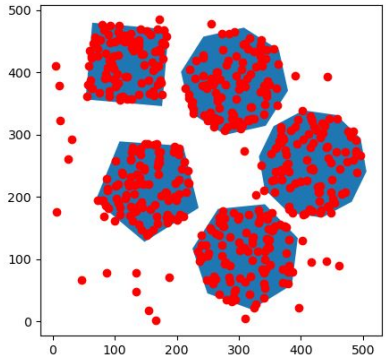


Fig. 2.4. 500 sample points with 34 noise points

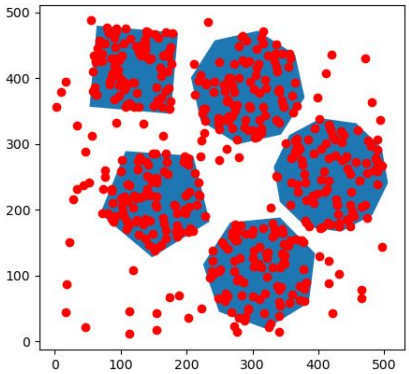


Fig. 2.5. 500 sample points with 66 noise points

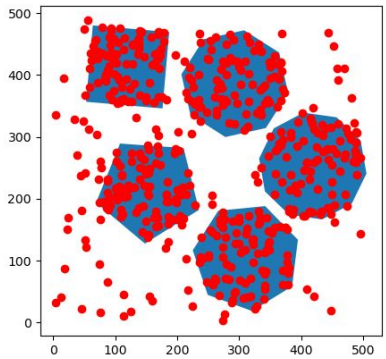


Fig. 2.6. 500 sample points with 82 noise points

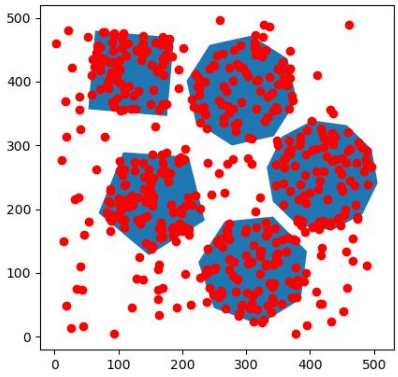


Fig. 2.7. 500 sample points with 99 noise points

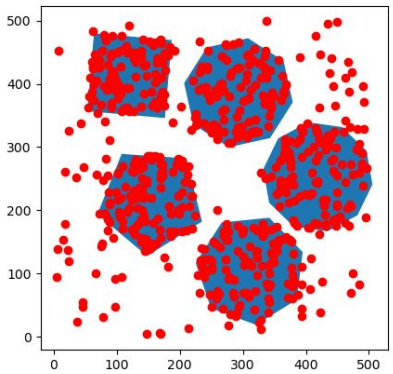


Fig. 2.8. 600 sample points with 99 noise points

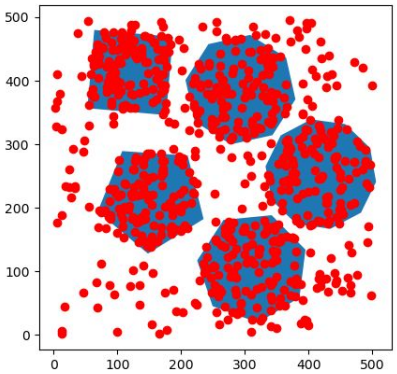


Fig. 2.9. 700 sample points with 179 noise points

- **DBSCAN**

DBSCAN is a density based unsupervised clustering algorithm. This algorithm does not need a number of clusters as the input. All it needs is the min radius and min points to calculate a core point. And it creates the clusters by connecting all the density reachable points in the dataset.

We ran the DBSCAN algorithm on iris data set on different radius parameters and min points. We found out that radius parameter is the most influencing parameter for this data set. Thus, we experimented on different radius values while keeping min point parameter to 6. As shown in the result table for smaller radius values, we got a lot of noise points and less clusters. When we increased the radius got more better answers. We also included the plot of Silhouette coefficients of each point against the sorted clusters.

Radius	Min points	Purity	Silhouette Coefficient	Truth clusters	Clusters	Noise
0.1	6	0.3333333	0.999997456	3	0	1
0.15	6	0.3733333	0.080176768	3	1	1
0.2	6	0.4466667	0.185186235	3	2	1
0.3	6	0.6266667	0.040847976	3	2	1
0.485	6	0.72	0.458814223	3	2	1
0.7	6	0.6733333	0.52507242	3	2	1
1	6	0.6666667	0.52507242	3	2	0

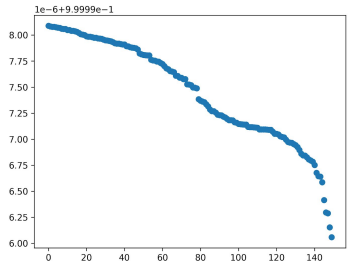


Figure 3.1.1

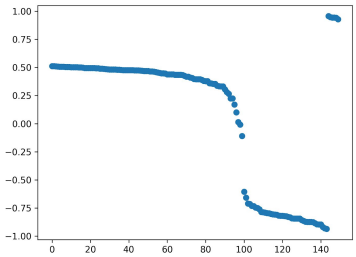


Figure 3.1.2

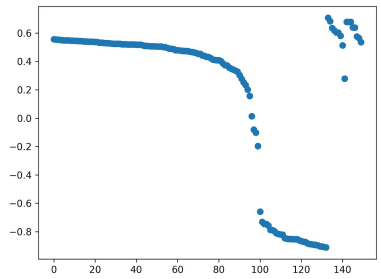


Figure 3.1.3

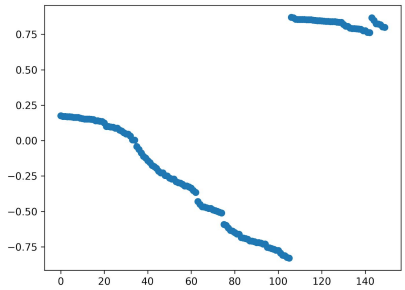


Figure 3.1.4

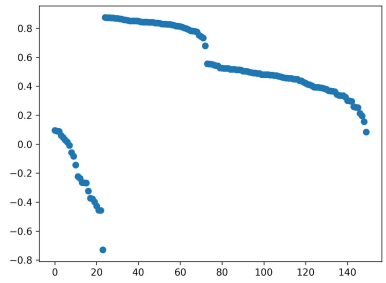


Figure 3.1.5

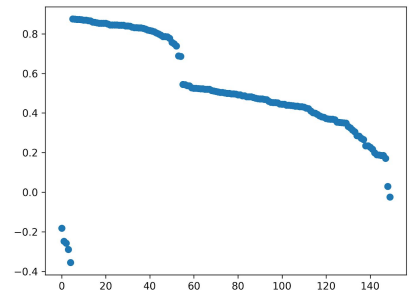


Figure 3.1.6

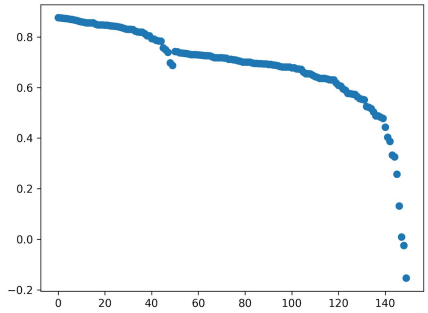


Figure 3.1.7

When we have radius 0.1 all the points are marked as noise. Thus we do not get a good silhouette coefficient. All the points have s.c 1. And purity value of 0.33. This is because all the points are marked as noise clusters. (We considered noise points as a cluster). However when we increase radius we get better purity and S.C values. As you can see for radius 0.285 we got the highest purity values that means the clusters we create have good precision, and we got a S.C of 0.45 as well. If we look at figure 3.5 we can see two clusters having good S.C values and one cluster with bad S.C values. (These are the noise points in the result). When we increase the radius to 0.7 we get a purity of 0.67 and S.C of 0.52. And when we increase the radius furthermore the number of noise points reduces to zero but we still only get two clusters from the result. Looking at the silhouette plots we can say the best results come when radius values are between 0.4 and 1. (Figure 3.1.5 , Figure 3.1.6, Figure 3.1.7) This makes us wonder whether DBSCAN is a good algorithm to use on this data set. For the DBSCAN algorithm it is very hard to capture the last ground truth cluster as a separate one. When we further investigated, we found that 3 ground truth clusters in the iris data set have different densities thus using a density based algorithm it is harder to capture all of these ground truth clusters. And since DBSCAN outputs the noise data points it becomes harder to interpret the result of DBSCAN using S.C. Our conclusion of this data set is, it is not good to use a density based algorithm to cluster this dataset. Further investigation revealed that we should have used an assessment score like F-score to measure the results for this algorithm.

Then we ran DBSCAN on a synthetic data set. We had 9 data sets with different sample size and noise points. Following table outputs the result.

File	Radius	Min points	Truth Clusters	Noise points	Clusters	Purity	Silhouette Coefficient
Synthetic_300S_63N.csv	35.5	4	5	1	5	0.76	0.409580931
Synthetic_400S_63N.csv	25.5	5	5	1	6	0.7125	0.099659453
Synthetic_500S_0N.csv	23.5	6	5	0	5	0.972	0.492354903
Synthetic_500S_34N.csv	26	5	5	1	4	0.752	0.413679125
Synthetic_500S_66N.csv	26	5	5	1	4	0.756	0.378995095
Synthetic_500S_82N.csv	25.7	4	5	1	6	0.812	0.332911263
Synthetic_500S_99N.csv	25.7	4	5	1	7	0.826	0.282922577
Synthetic_600S_99N.csv	24.7	6	5	1	5	0.725	0.236067282
Synthetic_700S_179N.csv	24.7	6	5	1	7	0.767143	0.320241904

For each data set we tried different radius values and different min point values and came up with the best possible result we could get. A graphical representation of these data sets were mentioned in previous sections. As you can see, when the densities of the clusters change and the number of noise points change, we had to change the parameters to get better results. For the third data set where we did not have any noise points, had the best result. We got a purity of 0.97. When there are no noise points it makes it easier to find a better result. If you look at the 7 and 9 th dataset we got 7 clusters as the result but we actually got good purity values. If you look at the figure 2.7 and 2.9 noise points could be identified as clusters. And also there are noise points in between ground truth clusters. This makes it harder to distinguish ground truth

clusters as two different clusters for the algorithm. Because noise points in between the clusters are identified as border points and connect the two clusters as one. So we can conclude that when we have noise points in between the borders of the clusters it becomes harder to identify them as noise points, we can certainly increase the min points in the algorithm parameter, but if we have different density clusters it would affect the results. In The data set 8 (figure 2.8) we got 5 clusters as the result, but our purity values are little bit low compared to the number of clusters, if you look at the figure 2.8 you can see the truth clusters in the right side bottom are very close and algorithm determined that these two clusters are same while making the noise points in the top right as a separate cluster. This is why we got a 0.72 purity value.

In the following plots we have included the silhouette coefficient values against sorted cluster lists. For each data set we tried to pick the best parameters by looking at the silhouette coefficient plots as well. As you can see most of the clusters in plots have good silhouette coefficients. We tried to pick the parameters such that the decrease of silhouette coefficient for each cluster is minimized. But when we have a lot of noise points we can see rapid decrease of silhouette coefficients. (Ex: Figure 3.2.9. Figure 3.2.2)

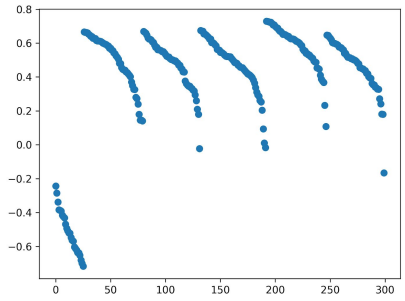


Figure 3.2.1

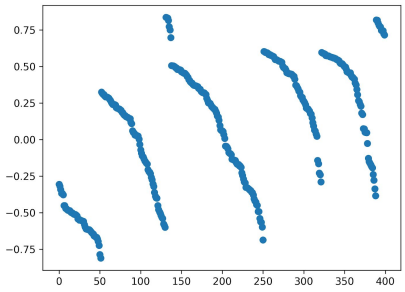


Figure 3.2.2

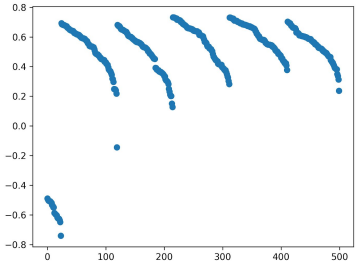


Figure 3.2.3

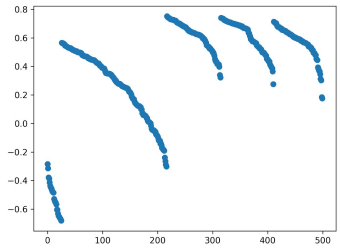


Figure 3.2.4

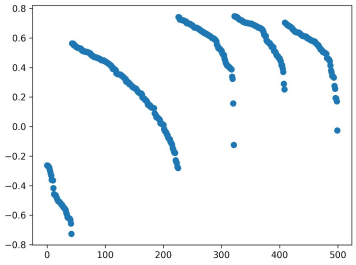


Figure 3.2.5

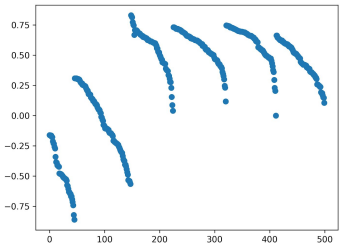


Figure 3.2.6

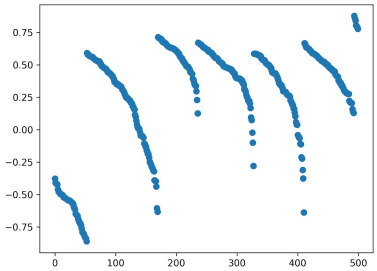


Figure 3.2.7

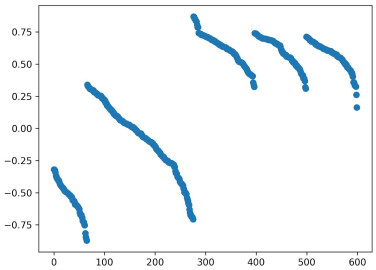


Figure 3.2.8

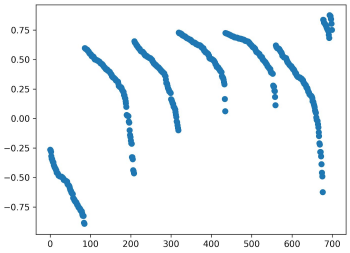


Figure 3.2.9

- **K- means**

K-means is a representation based unsupervised clustering technique. With a user given a predetermined number of clusters(K), the algorithm tries to assign the data points to the nearest cluster center. Cluster centers are initially assigned randomly and in each iteration, after the points are assigned to the nearest center the centers are updated to be the new mean of the assigned data points within the clusters. Iterations

are continued until the sum of squares of cluster movement is smaller than a given parameter(epsilon).

Due to the random nature of the initialization, the final results greatly depend on the initial conditions. Which could trap the algorithm in local minima rather than achieving a global minimum. To mitigate this problem several epochs are run with the same parameters and find the best performing epoch by an internal assessment metric.

In order to improve the initialization, the K-means ++ algorithm is used. Which places the initial cluster centers smartly rather than a completely random process. The first cluster center is placed randomly within the data input range. Then the consecutive cluster centers are chosen to be the furthest data point from the previous cluster centers. This will allow the cluster centers to be spread apart so that they have more freedom to acquire data points.

The script runs the k-means algorithm with a predefined range of parameters of cluster numbers(K) and error threshold(epsilon) and finds the best parameters with the best internal metric.

The K-means algorithm is applied to both iris dataset and the synthetic dataset.

For the Iris dataset the *epsilon* value was fixed at 0.01 as it does not affect the results much. The silhouette coefficient is calculated for different *K* values. The following table shows the results for Iris dataset.

The Iris data results with *epsilon = 0.01*

# of clusters (K)	Silhouette coefficient	Purity
2	0.68081362	0.666666667
3	0.552591945	0.893333333
4	0.49782569	0.88
5	0.493129486	0.873333333
6	0.448306506	0.906666667

The silhouette coefficients for each point for each number of clusters can be plotted as below.

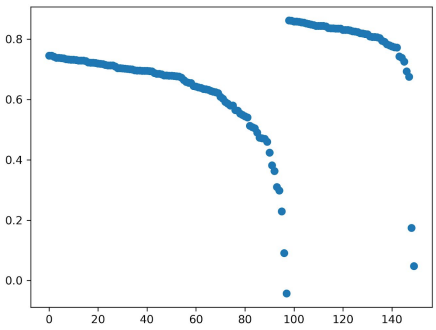


Fig 4.1: $K = 2$

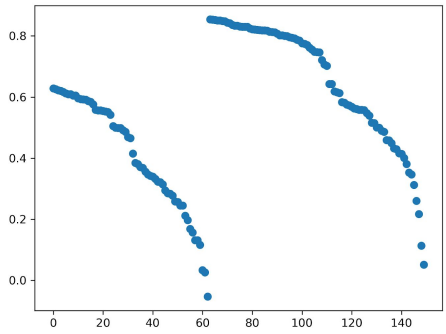


Fig 4.2 : $K = 3$

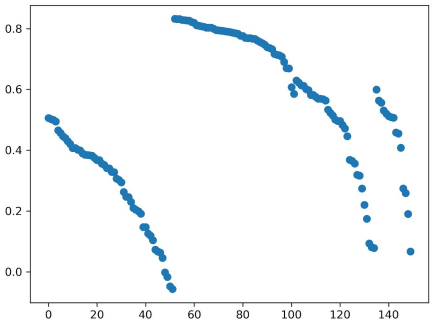


Fig 4.3: $K = 4$

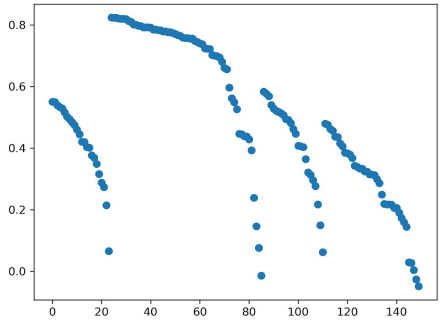


Fig 4.4: $K = 5$

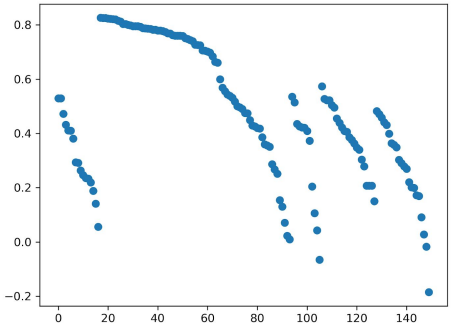


Fig 4.5: $K = 6$

As per the results the $K=2$ gives the best results although the iris data has 3 classes. The $K = 2$ case gives the highest average silhouette coefficient. Even though $K = 3$ has higher values for a single cluster the average is less than the $K = 2$ case.

For the synthetic data the number of classes is kept fixed at 5 and the number of class samples and noise samples is varied. The experiments were conducted with fixed *epsilon* = 1 as it does not affect the results much. The best *K* is found by searching through the values 4 to 10 with *epochs* = 3 for each run.

The synthetic data results with best parameters

File	# of best clusters	Purity	Silhouette Coefficient
Synthetic_300S_63N.csv	5	0.68	0.45778505346869725
Synthetic_400S_63N.csv	6	0.81	0.45538474794041156
Synthetic_500S_0N.csv	5	1.0	0.5366440721770801
Synthetic_500S_34N.csv	5	0.898	0.5093693000884791
Synthetic_500S_66N.csv	5	0.828	0.4881397278181793
Synthetic_500S_82N.csv	6	0.79	0.4865309078502208
Synthetic_500S_99N.csv	5	0.738	0.4687924936495102
Synthetic_600S_99N.csv	5	0.765	0.47393470256032677
Synthetic_700S_179N.csv	5	0.62	0.4481789746347287

The silhouette values for the best hyper parameters can be plotted as follows, with Nc = number of cluster points and Nn = number of noise poins.

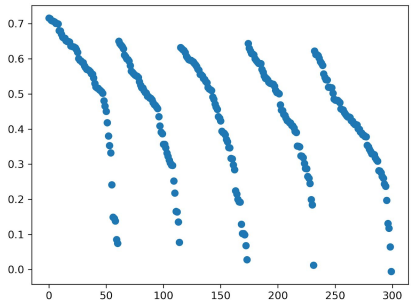


Fig 4.6: Nc = 300, Nn = 63

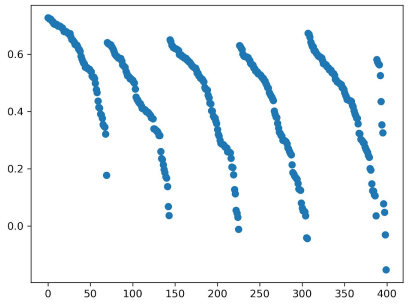


Fig 4.7: Nc = 400, Nn = 63

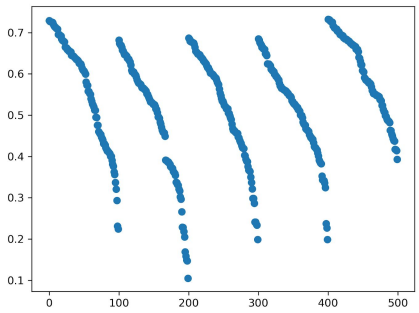


Fig 4.6: Nc = 500, Nn = 0

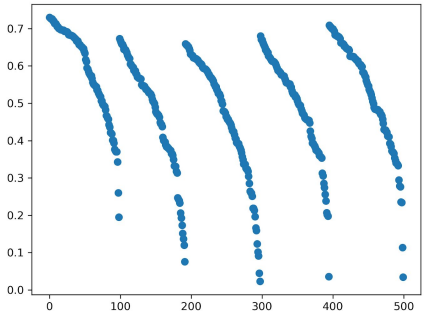


Fig 4.7: Nc = 500, Nn = 34

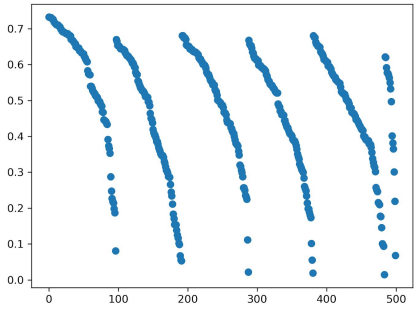


Fig 4.6: Nc = 500, Nn = 66

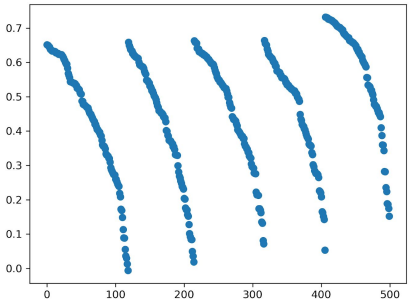


Fig 4.7: Nc = 500, Nn = 82

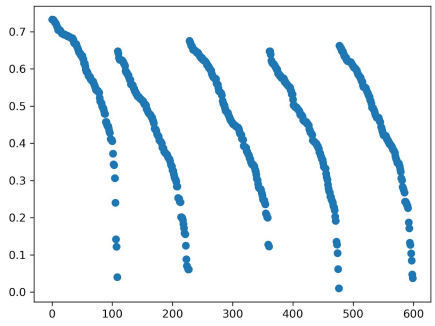


Fig 4.6: Nc = 600, Nn = 99

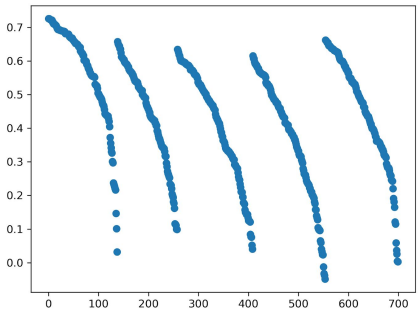


Fig 4.7: Nc = 700, Nn = 179

The K -means algorithm doesn't have the ability to distinguish between noise points and cluster points. So it will assign all the noise points to the clusters as well. Hence the purity will decrease with the introduction of the noise points. As the noise increases the purity decreases. Also since in the synthetic data two clusters are close together they will be classified as a one cluster. It is hard to separate them by just

evaluating the silhouette coefficient. Hence, evaluating using different internal assessment metrics will be beneficial.

- **Conclusion**

Our understanding is that when the algorithm outputs noise points in the result silhouette coefficient becomes harder to interpret, thus it is better to use external assessment metrics for these algorithms. And we believe we should have used a metric like F-score to determine that. For the iris data set, it is better to use k-means algorithm, mostly because the clusters in the iris data set have different densities, thus it is harder for DBSCAN to catch those clusters. For the synthetic data set we believe DBSCAN outputs better results, although it depends on the cluster densities and percentage of noise points. Assuming that clusters have similar densities DBSCAN outputs pretty good results when the percentage of noise points is low. The problem with noise points arises when noise points reside between ground truth clusters, where the DBSCAN algorithm labels them as border points. If noise points between clusters have a higher distance to cluster borders DBSCAN can label them as outliers (noise).