Unsupervised Learning - project

For this project, I have chosen to implement the following clustering algorithms:

- DBSCAN
- Spectral clustering
- Agglomerative clustering

All code files are attached in the folder, including the test.py file which contains the code that tests these algorithms on several datasets and plots the results. The code uses the following python packages:

- *Numpy*: for calculations and data management
- *Scipy*: for calculations
- *Matplolib.pyplot*: presents the different plots
- *Sklearn.cluster*: compares the results of existing implementations of these algorithms
- *Sklearn.datasets*: creates the data to apply the algorithms

I have chosen these algorithms because each algorithm individually presents interesting and unique approaches to handling and clustering data. Each algorithm has its own advantages and disadvantages, which will elaborated on in this project.

To test these algorithms, I have used 2D datasets in order to visualize the results. In addition, I used the MNIST dataset to measure the success rate of the algorithms.

DBSCAN:

"Density Based Spatial Clustering of Applications with Noise". This method clusters the data with respect to the density of the data. It takes two arguments: epsilon and minimum points. The first stage is to classify all data points to 3 categories:

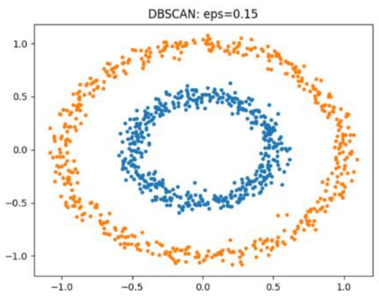
- Core Points: data points that has at least minimum points within radius of epsilon.
- Border Points: data points that has less than minimum points but more than 0 points within radius of epsilon.
- Noise: points that has 0 data points within radius of epsilon.

After classifying the points, the algorithm starts from a random core point and form a cluster. Each data point that accessible from the core point is added to the cluster. As a result, the subsequent noise will not be added to any cluster.

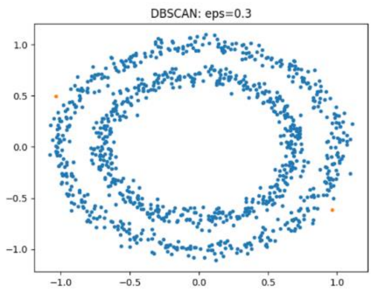
This method is very efficient and quick. It can identify arbitrarily shaped clusters and does not need to take the number of clusters (k) like other clustering methods (for example: k-means). However, sometimes it is hard to guess a good epsilon or minimum points, and different values might end up with different results.

For example, let's take two datasets and preform the algorithm with different values:

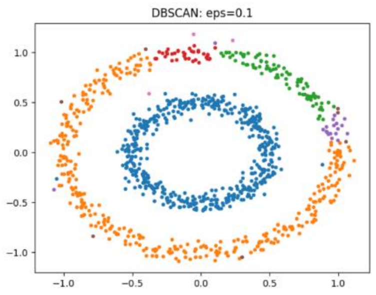
Eps = 0.15:



Eps=0.25:

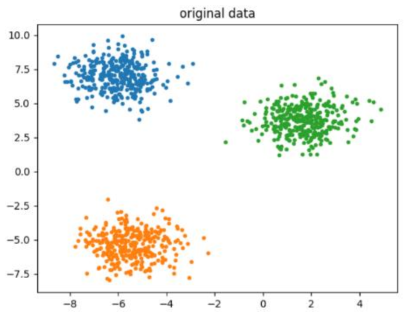


Eps=0.1

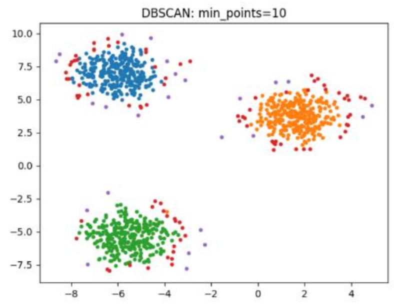


We can see that for eps=0.15, DBSCAN is able to cluster the data to the correct circles, but for eps=0.3 it fails because the radius is too large, and all data points are in a single cluster. Eps=0.1 also fails because it is too small, and we get 5 clusters. Here, the minimum points are the same for all 3 epsilon values.

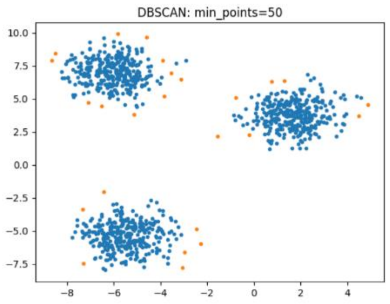
original



minimum points=10

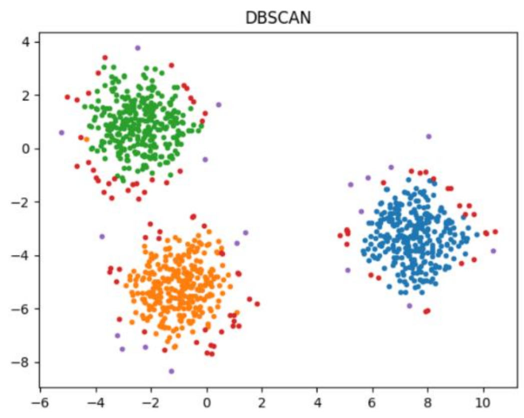


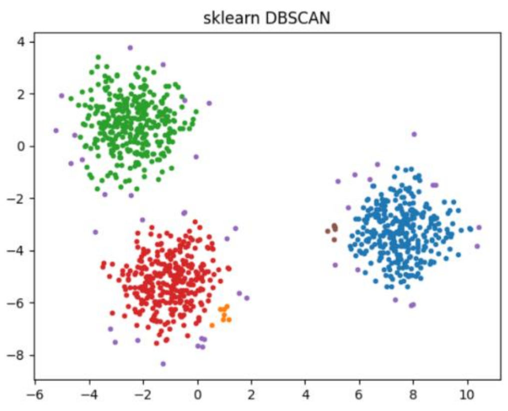
minimum points=50



In this case, we can see that the original data is 3 different groups. With minimum points=10 we get good results of 3 clusters and noise around them (the red dots). With minimum, points=50 we get 1 cluster and little noise (orange dots) which is wrong. Here, the epsilon value is 0.5 for both tries.

Sklearn implementation of DBSCAN return similar results with the same values of epsilon and for the minimum points. For example:





<u>Agglomerative clustering:</u>

In this method, the algorithm refers to each data point as a cluster. The goal for each iteration is to find the pair of closest clusters and merge them into one. There are few options to measure the distance between clusters. I chose to implement two options:

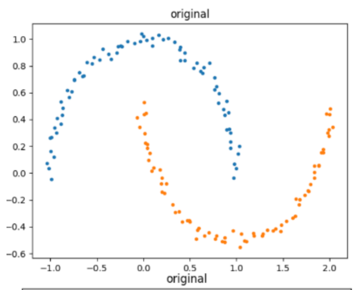
- Maximum Distance: The distance between clusters is the maximum Euclidian distance between two data points.
- Ward's method: First, we need to combine the clusters and find the mean of the new cluster. Then we compute the distance of each vector from the mean. The sum of those distances measured as the distance between the clusters.

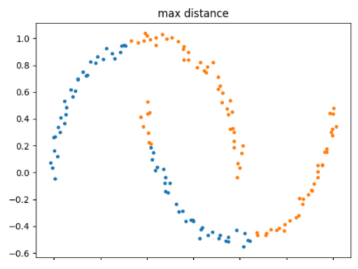
The algorithm stops when it reaches on cluster of all the data, or when it reaches to k clusters if specified.

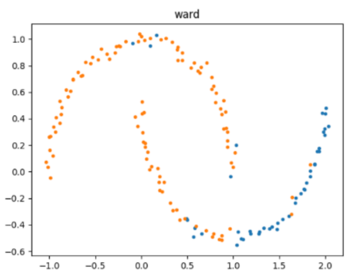
Agglomerative clustering is easy to understand and does not require to specify the number of clusters beforehand. In addition, the algorithm will always return the same results over the data (for the same distance method). However, it is not efficient, and for large amounts of data it takes a lot of time to finish the process ($O(n^3)$ is best time complexity of the algorithm).

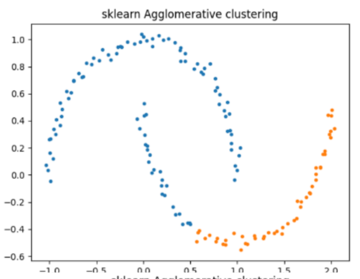
Let's examine the results of the algorithm over datasets with Ward's method and max distance method:

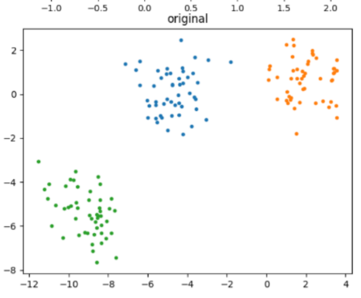
<u>Original</u> <u>Maximum Distance</u> <u>Ward's Method</u> <u>Sklearn</u>

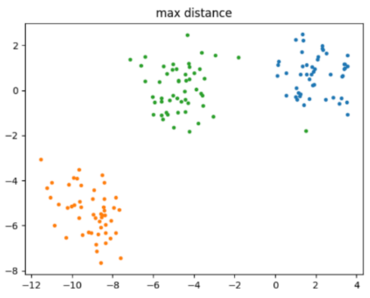


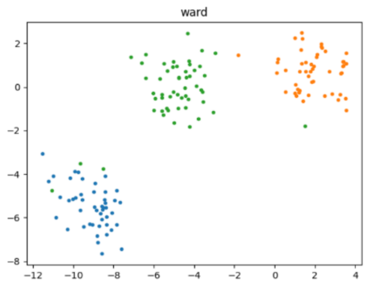


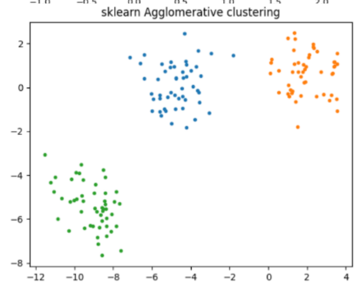












The Sklearn's implementation with default settings returns very similar results to my implementation using Ward's method. However, Sklearn's algorithm runs faster.

<u>**Spectral Clustering**</u>:

This algorithm forms an adjacency matrix from the data, then clusters the data using the eigenvectors of the Laplacian.

The adjacency matrix (also called similarity matrix) is achieved by referring to each data point as a node in a graph. For each two nodes (data points), $A_{i,j}$ represents the weight of the edge between them (provided the edge exists). There are two ways to form the adjacency matrix:

- RBF weights: In this method, $A_{i,j} = e^{-\left(\frac{\|x_i - x_j\|^2}{2 \cdot \sigma^2}\right)}$. Since the RBF deceases when $X_i \sim X_j$, it is good similarity measure.

- K-Nearst Neighbors graph: Here we find the closest K data points (by some vector norm), and then $A_{i,j} = \begin{cases} 1, & X_j \in K\_nearest\_neighbors(X_i) \\ 0, & otherwise \end{cases}$.
  (K is not related to the k clusters).

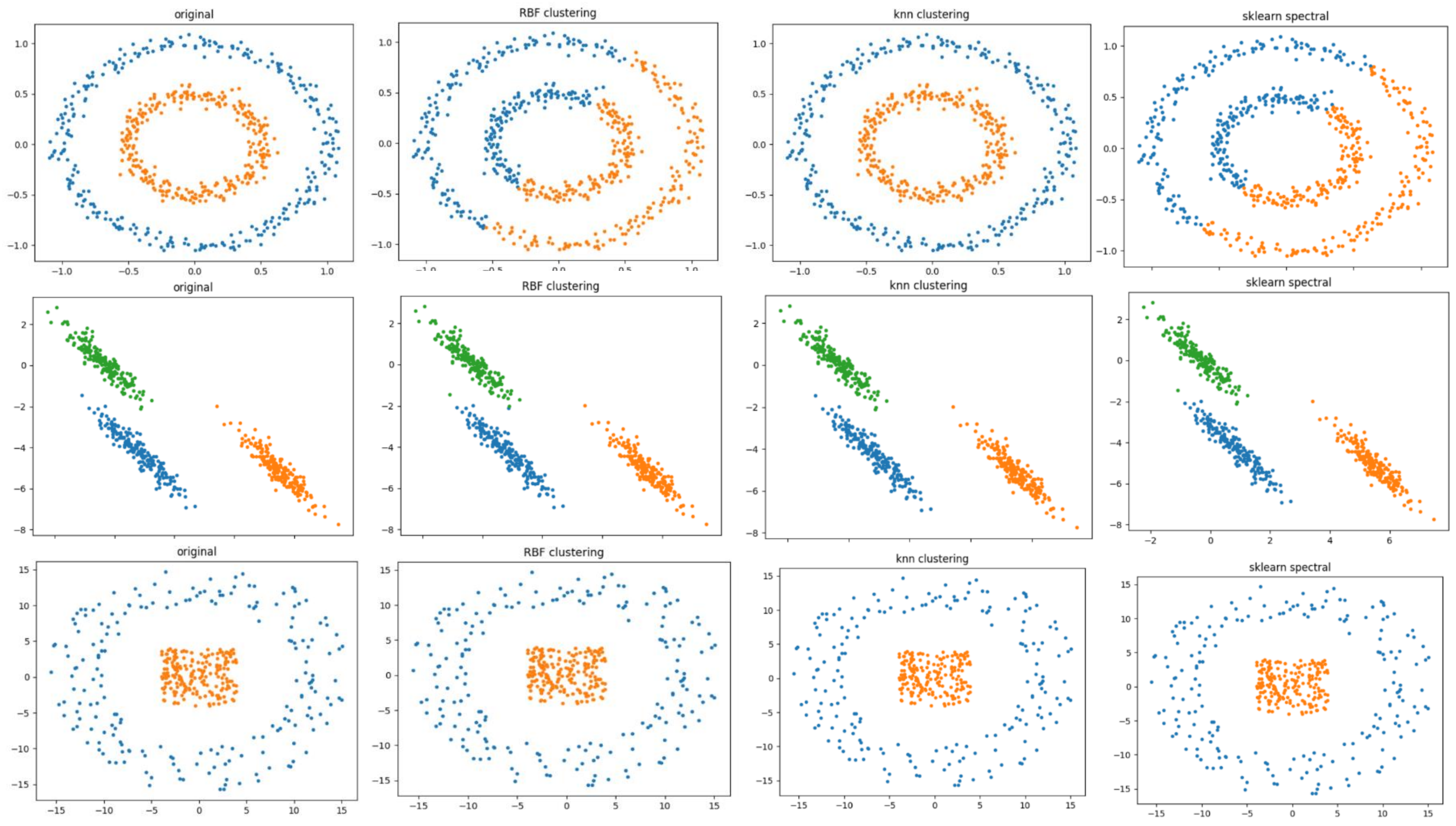
We can see that in both cases, the adjacency matrix is symmetric.

Then, we calculate the Degree matrix, which holds the sum of each row of the adjacency matrix on the diagonal, and 0 anywhere else. Now it is simple to find the Laplacian, using the formula $L = D - A$, where D is the degree matrix and A is the similarity matrix. The next step is to find the corresponding eigenvectors to the smallest k eigenvalues and form them to a matrix where row i represents X_i. Finally, we can cluster this matrix using k-means (or any other clustering method) and can return the results.

Spectral Clustering is fast and is able to cluster different shapes and sizes. However, it uses $O(n^2)$ memory, and for large amounts of data it may be hard to calculate the eigenvectors. It is also dependent on the relevant clustering method we use over the eigenvectors.

Examples:

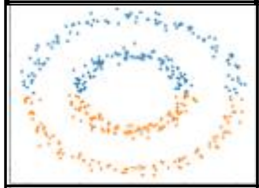
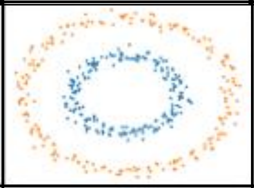
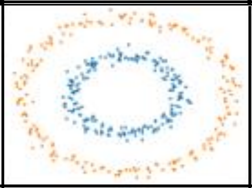
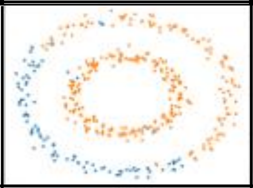
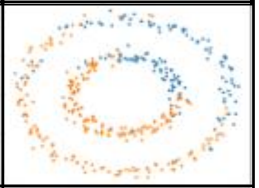
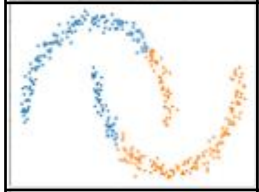
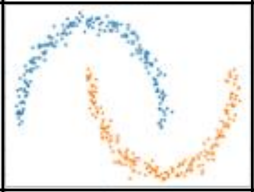
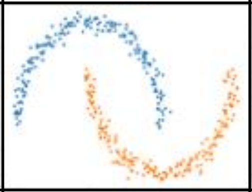
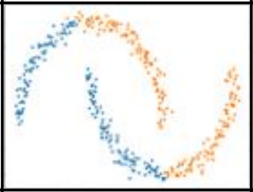
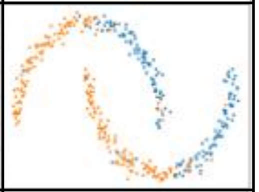
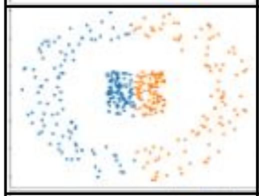
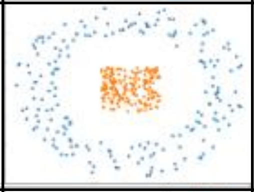
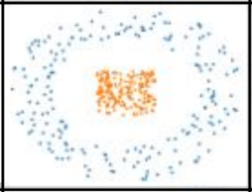
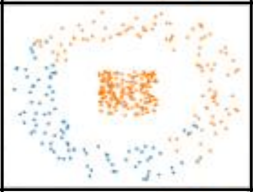
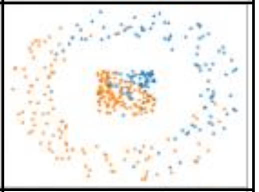
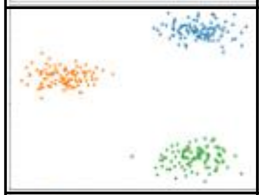
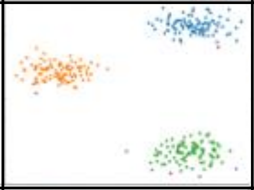
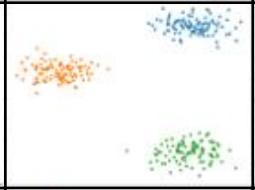
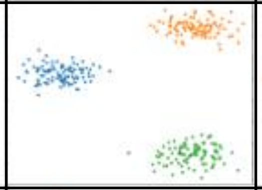
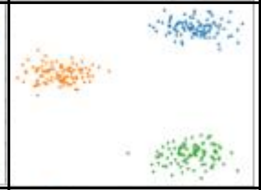
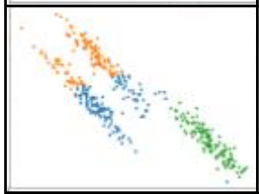
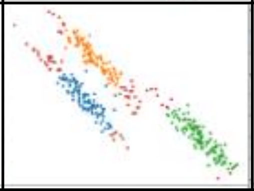
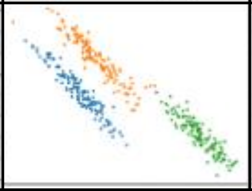
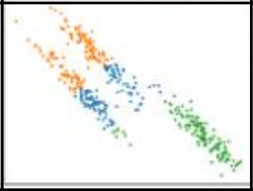
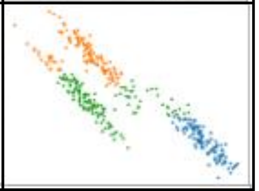
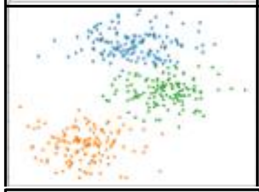
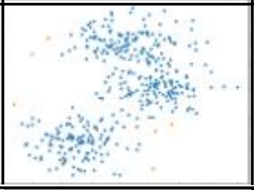
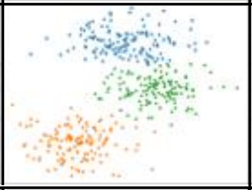
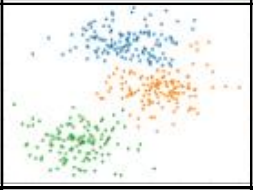
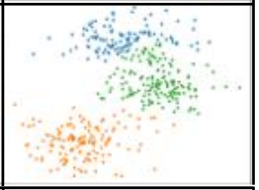
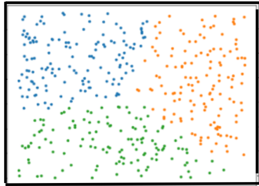
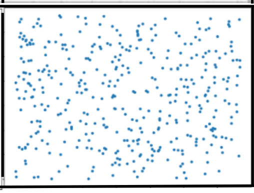
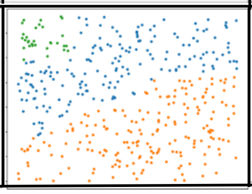
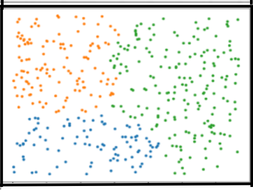
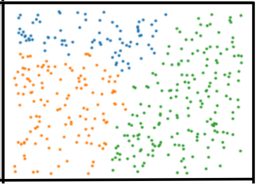
Original RBF matrix KNN matrix Sklearn



Sklearns's implementation with default settings have returned similar results to the RBF results from my algorithm.

<u>Side by side comparison between the algorithms:</u>

Let's see how the algorithms preform over the same data. I have also included K-means algorithm in this comparison to see the differences between them:

K-means	DBSCAN	Spectral Clustering	Agglomerative Clustering: Max Method	Agglomerative Clustering: Ward's method
				
				
				
				
				
				
				

MNIST test:

In this section, I have tested the algorithm over the MNIST data. In each test, all images of three different digits are taken form the first 500 digits in the training dataset. Then, apply each algorithm over those digits and get the labels. From the labels we find the cluster mean and the most common digit in the cluster, use this information to guess the label of an image from the test data. The accuracy of each algorithm and set of digits is in the table below:

Digits	**Spectral clustering**	**DBSCAN**	**Agglomerative Clustering**
1, 5, 6	45.16%	67.74%	45.16%
0, 3, 7	56.61%	61.02%	55.14%
2, 8, 9	51.06%	53.02%	51.06%
0, 1, 8	65.77%	64.42%	73.15%
1, 3, 6	60.64%	63.87%	63.87%

We can see that for this task, in general DBSCAN performed better the spectral clustering and agglomerative clustering.

Tomer Garbi, 208956706