SPR Project Presentation

Clustering Methods

Jayanth S 201081003

Praveen Kumar N 201082001

Rishabh Roy 201082002

Introduction

- Clustering is a type of unsupervised learning method in machine learning. It is a task of dividing the data sets into a certain number of clusters in such a manner that the data points belonging to a cluster have similar characteristics.
- Types of clustering algorithms:
  1. *Partitioning* algorithms (Ex : k-means, Partitioning Around Medoids(PAM)).
  2. *Hierarchical* algorithms (Agglomerative and divisive approach).
  3. *Density based* clustering(Ex: DBSCAN, HDBSCAN).
  4. *Fuzzy* clustering(soft clustering)(Ex: fuzzy c-means).

- Clustering is a type of unsupervised learning method in machine learning.
- Clustering divides the data sets into a certain number of clusters in such a manner that the data points belonging to a cluster have similar characteristics.
- Partitioning algorithms partitions a dataset D of n points into a set of k clusters, k is an input parameter for these algorithms,which requires some domain knowledge.(Ex: *k-means*, *Partitioning Around Medoids(PAM)*).
- Hierarchical algorithms create a hierarchical decomposition of D represented by a dendrogram.

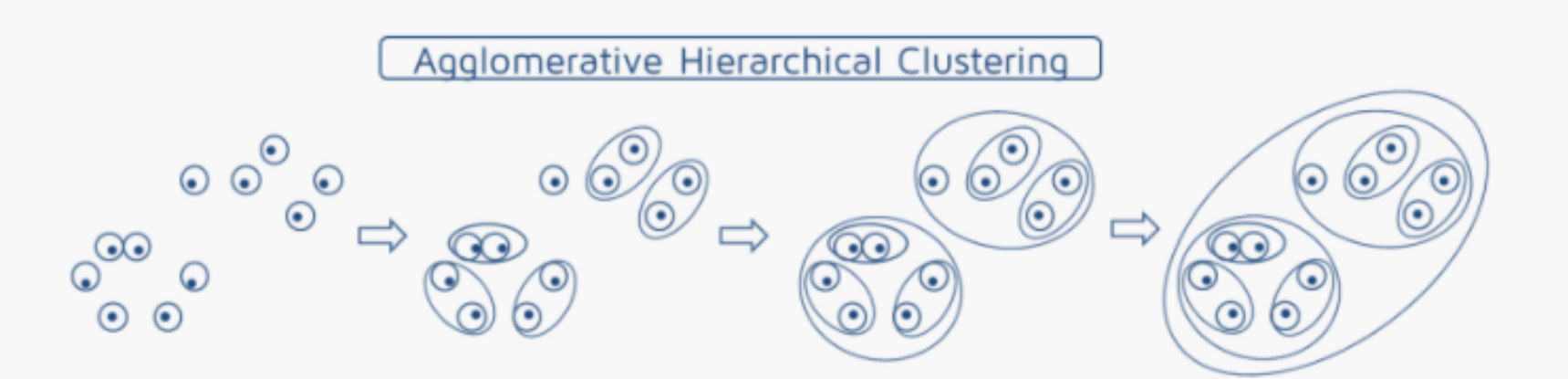
Hierarchical Clustering

- Hierarchical clustering groups the given data into a tree of clusters which helps in data summarization and visualization.

Figure: Dendrogram that helps in representing the results of hierarchical clustering

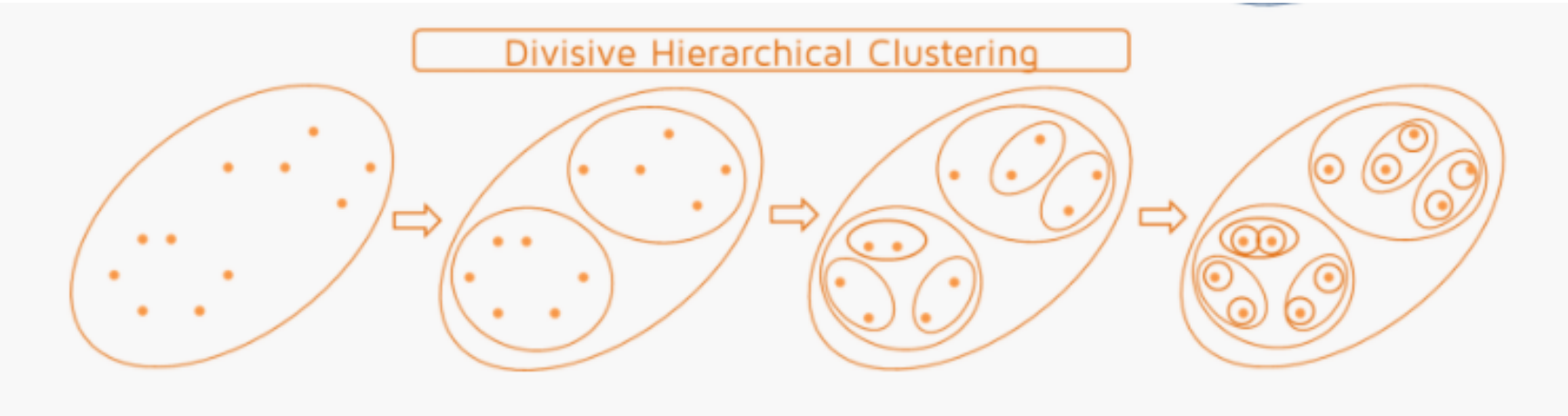
Agglomerative Clustering

- This is a bottom-up approach, where we consider each sample as a cluster.
- Then we start aggregating the clusters successively until all the samples fall into a single cluster. If two data points appear together in same cluster then they can't be separated in the upcoming steps.



Divisive Clustering

- This is a top-down approach in which we start by considering all the samples as a single cluster. Then we the split the clusters at each step.



Basis on which Clusters are Combined or Split

- We combine different clusters at each stage using
  - Appropriate Metric
  - Linkage Criterion

Metrics that are used in Hierarchical clustering

- Let a $\in \mathbb{R}^d$,b $\in \mathbb{R}^d$ be two points. Then, commonly used metrics are
  - Squared Euclidean distance

$$\|a - b\|_2^2 = \sum_{i=1}^{d} (a_i - b_i)^2$$

  - Manhattan distance : It is the absolute distance between two vectors. (Also called as l_1 norm)

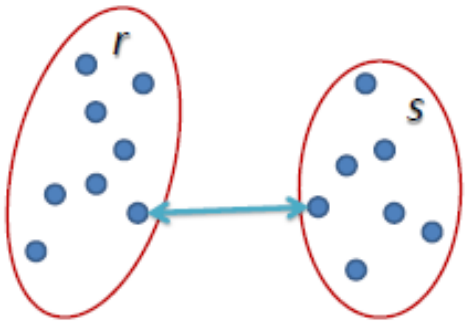
$$\|a - b\|_1 = \sum_{i=1}^{d} |a_i - b_i|$$

  - Maximum distance : It is the maximum distance between two components of a and b.

Linkage Criterion

- Let $\mathcal{D}_i$ and $\mathcal{D}_j$ be 2 different clusters. The linkage criterion functions that are used are,
  - Single-linkage criterion (Nearest Neighbor clustering) : The 2 samples (one in each cluster) that are closest to each other (most similar pairs in 2 clusters) will determine the distance between two clusters.

$$d_{min}(\mathcal{D}_i, \mathcal{D}_j) = \min_{x \in \mathcal{D}_i, x' \in \mathcal{D}_j} \|x - x'\|$$

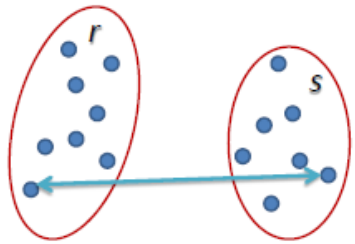


$$L(r,s) = \min(D(x_{ri}, x_{sj}))$$

Linkage Criterion

- Complete-linkage criterion (Farthest Neighbor Clustering) : The 2 samples (one in each cluster) that are farthest to each other (least similar pair between two clusters) will determine the distance between two clusters.

$$d_{max}(\mathcal{D}_i, \mathcal{D}_j) = \max_{x \in \mathcal{D}_i, x' \in \mathcal{D}_j} \|x - x'\|$$



$$L(r,s) = \max(D(x_{ri}, x_{sj}))$$

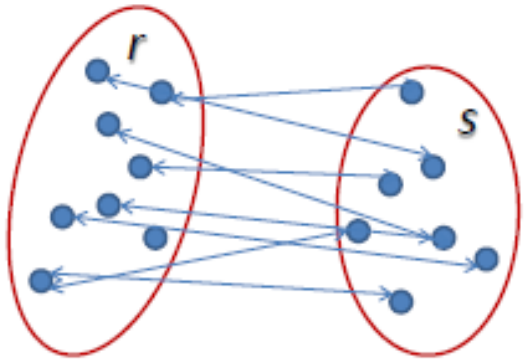
Figure: Complete linkage

Linkage Criterion

- Average-linkage criterion :

  The distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster.

$$d_{avg}(\mathcal{D}_i, \mathcal{D}_j) = \frac{1}{n_i n_j} \sum_{x \in \mathcal{D}_i} \sum_{x' \in \mathcal{D}_j} \|x - x'\|$$



$$L(r,s) = \frac{1}{} \sum^{n_r} \sum^{n_s} D(x_{ri}, x_{sj})$$

Linkage Criterion

- Ward-mimimum variance criterion :
  Ward's minimum variance criterion minimizes the total within-cluster variance.
- At each we merge the pair of clusters that leads to minimum increase in total within-cluster variance after merging. This increase is a weighted squared distance between cluster centers.

$$d(C_i \cup C_j, C_k) = \frac{n_i + n_k}{n_i + n_j + n_k} d(C_i, C_k) + \frac{n_j + n_k}{n_i + n_j + n_k} d(C_j, C_k) - \frac{n_k}{n_i + n_j + n_k} d(C_i, C_j).$$

Agglomerative Clustering

- Given $\mathbb{D} = \{X_i\}_{i=1}^{n}$
- Choose the distance metric for a pair of points and linkage criterion for pair of clusters i.e., $d(\mathcal{D}_i, \mathcal{D}_j)$.
- Start from n clusters and at each iteration :
  - Find the nearest clusters and merge them.
  - Do the above step until the number of clusters is 1.
- If we know the specific number of clusters to be formed ($\overline{k}$) for the given samples, then we can stop when number of clusters is equal to $\overline{k}$.

- Pros : No need to specify the number of clusters in hierarchical clustering.
- Cons :
  - It may not provide the best solution always.
  - When the feature vectors of sample data contain different data types, computing the distance between different clusters at each stage is difficult.

- When Linkage criterion is average linkage and distance metric is euclidean

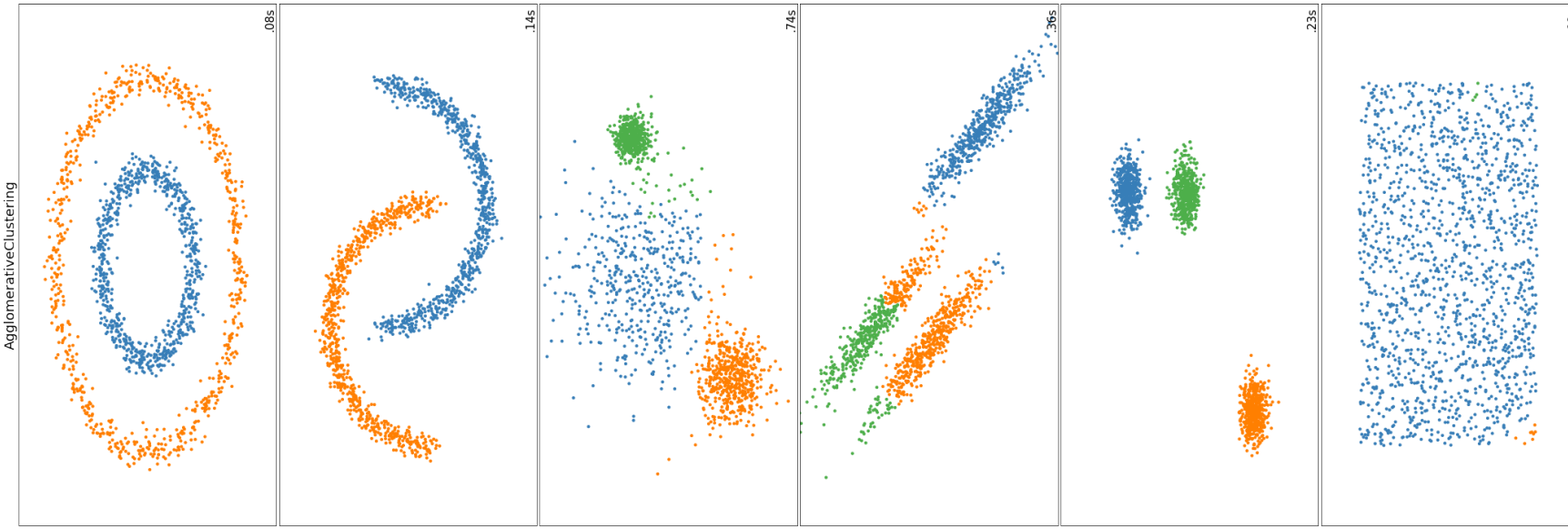


Figure: Linkage Criterion : Average Linkage, Metric : Euclidean

- When Linkage criterion is average linkage and distance metric is *l*1 norm

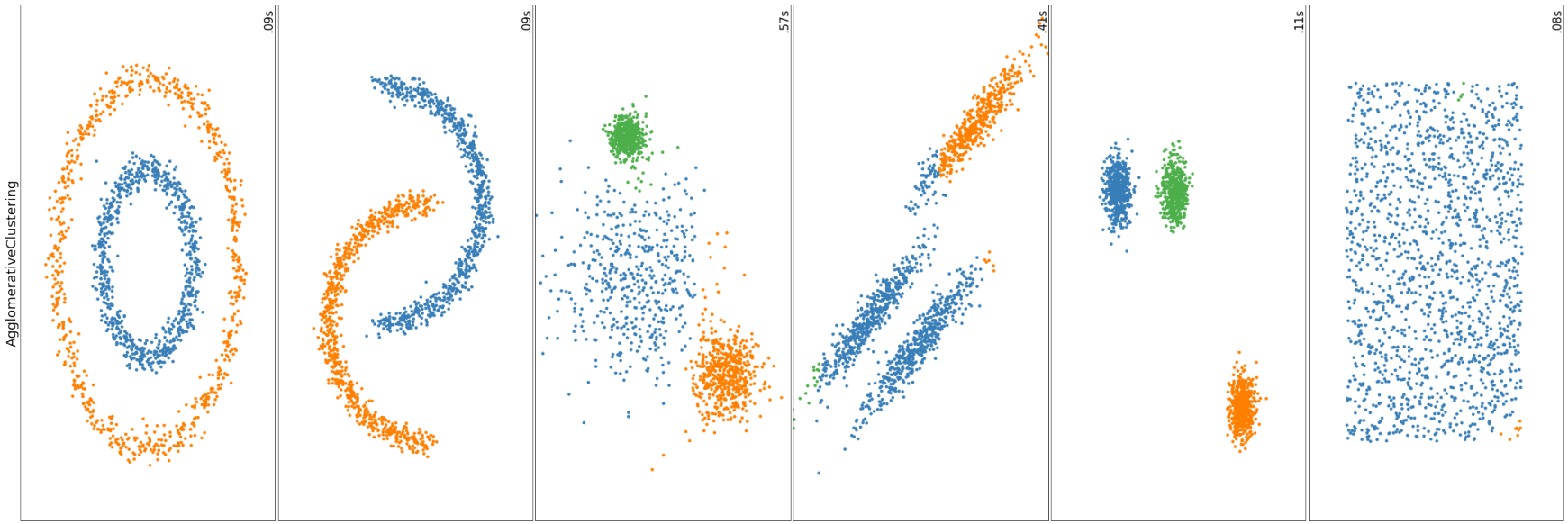


Figure: Linkage Criterion : Average Linkage, Metric : L1 Norm

- When Linkage criterion is single linkage and distance metric is euclidean

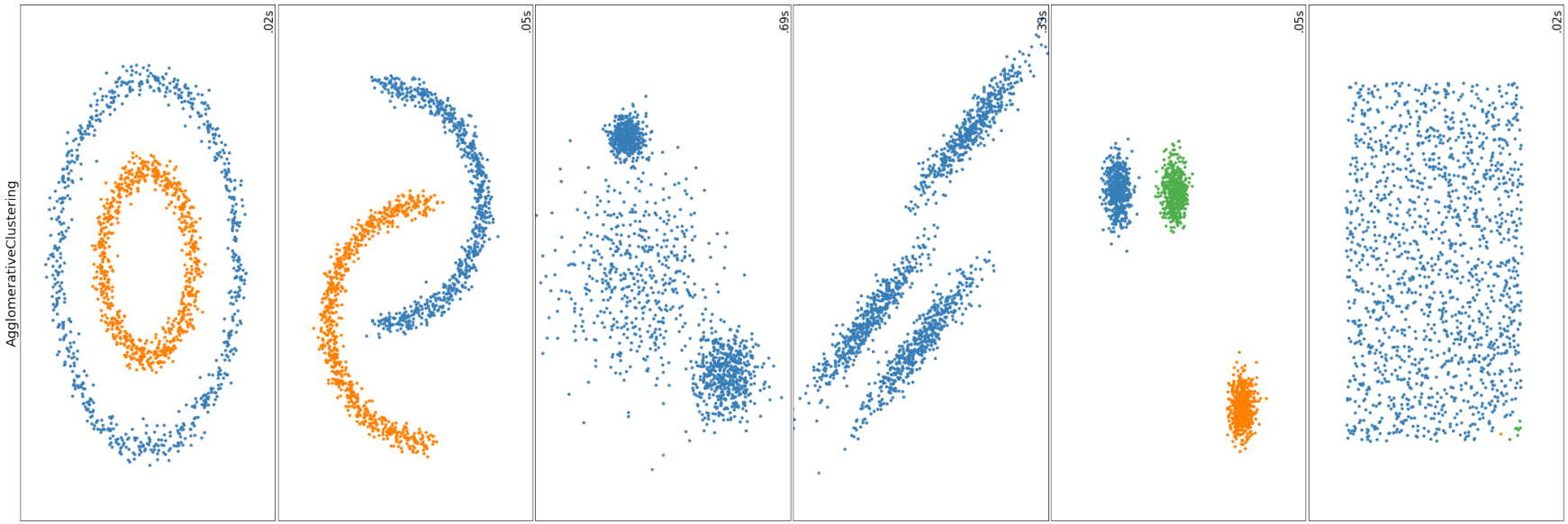


Figure: Linkage Criterion : Single Linkage, Metric : Euclidean

- When Linkage criterion is single linkage and distance metric is *l*1 norm.

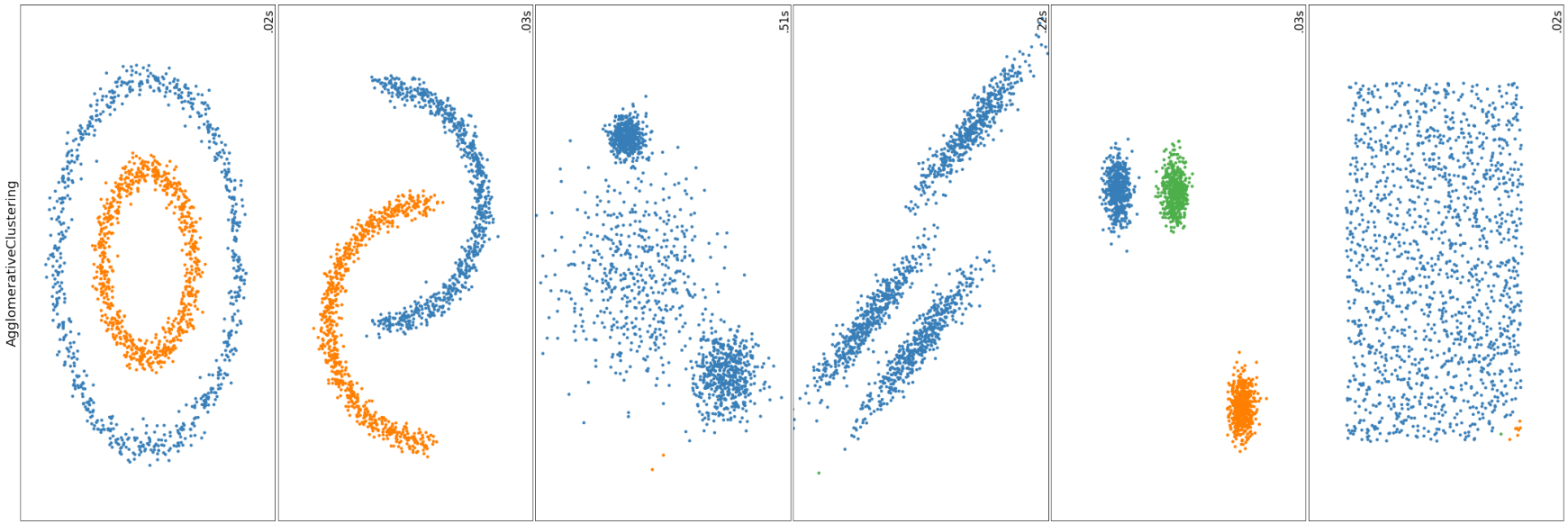


Figure: Linkage Criterion : Single Linkage, Metric : L1 norm

- When Linkage criterion is single linkage and distance metric is euclidean

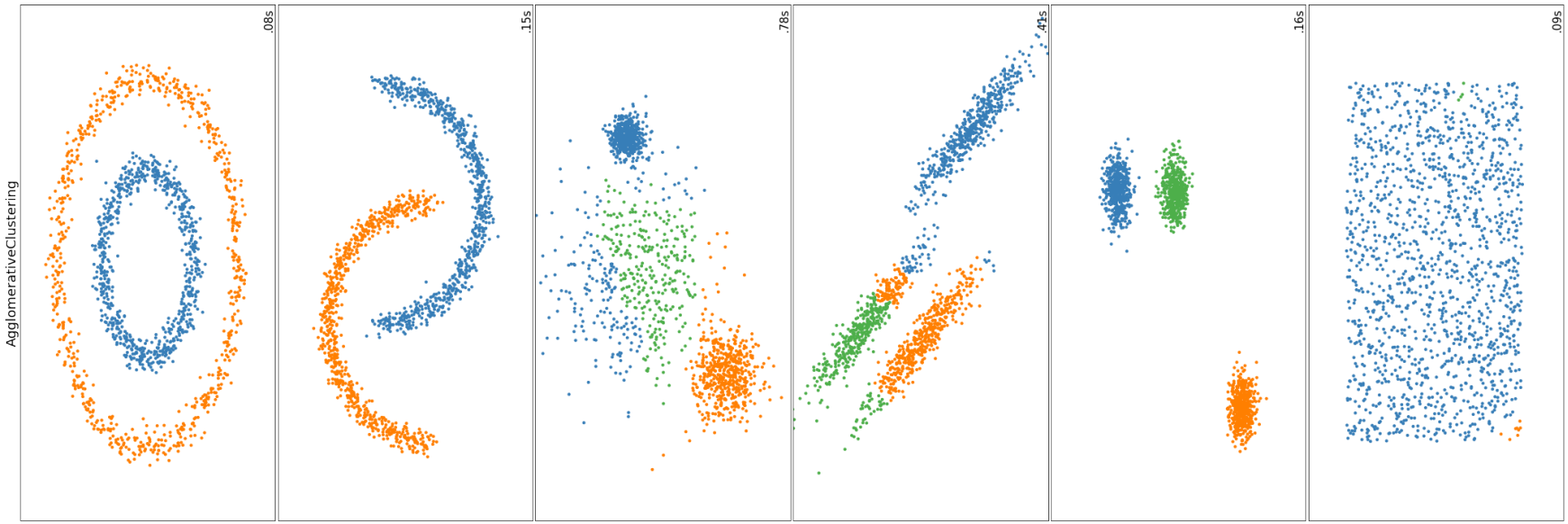


Figure: Linkage Criterion : Complete Linkage, Metric : Euclidean

- When Linkage criterion is complete linkage and distance metric is *l*1 norm.

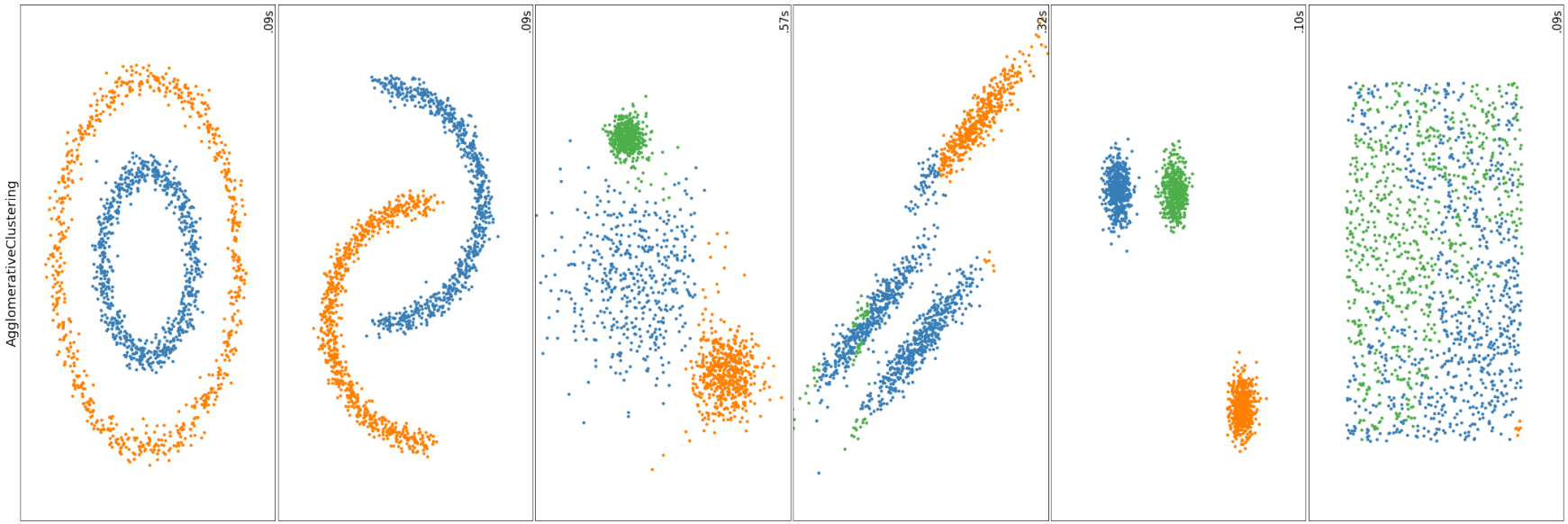


Figure: Linkage Criterion : Complete Linkage, Metric : L1 norm

- When Linkage criterion is Ward minimum variance linkage and distance metric is euclidean norm.

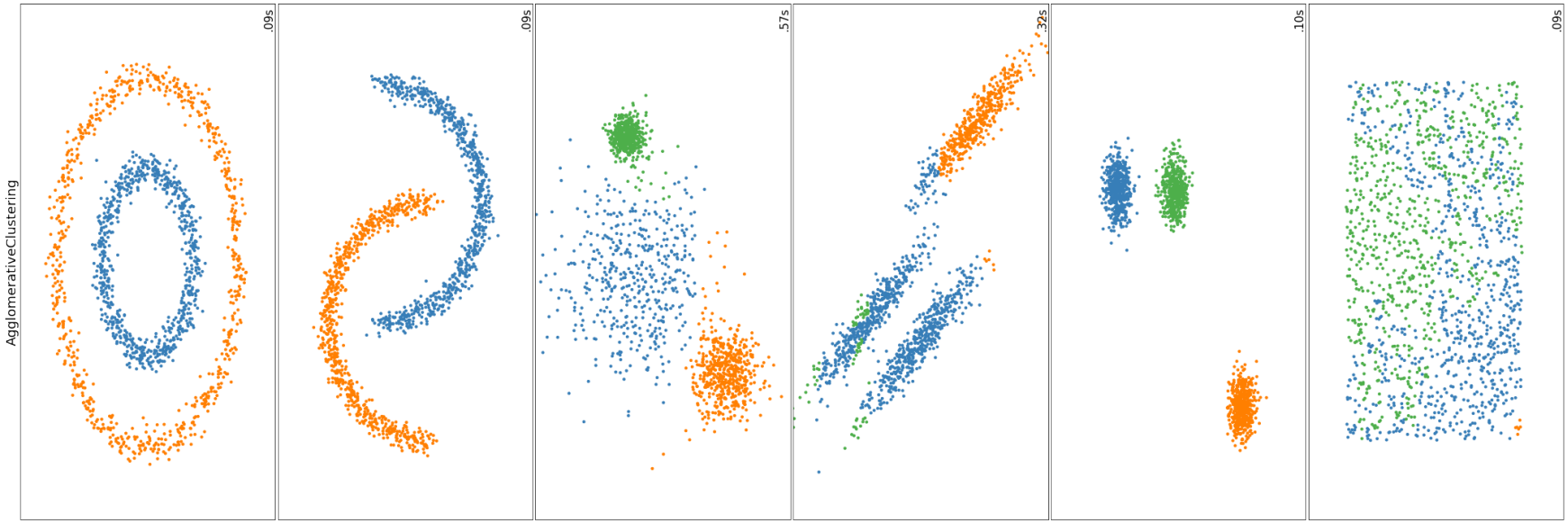


Figure: Linkage Criterion : Ward minimum variance linkage, Metric : Euclidean norm

DBSCAN

- While dealing with spatial clusters of different density, size and shape, it could be challenging to detect the cluster of points.
- The task can be even more complicated if the data contains noise and outliers. To deal with large spatial datasets,Density-Based Spatial Clustering of Applications with Noise (**DBSCAN**) was proposed.
- The main concept of DBSCAN algorithm is to locate regions of high density that are separated from one another by regions of low density.

DBSCAN Parameters

- In the case of DBSCAN, instead of guessing the number of clusters,we will define two hyperparameters: Eps(ϵ) and MinPts to arrive at clusters.
  - ϵ(**Eps or Epsilon**): The radius of the circle to be created around each data point to check the density.
  - **ϵ-nieghborhood**: ϵ-neighborhood of a point p denoted by $N_\epsilon(p)$, is defined as $N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\}$.
  - **MinPts**: The density threshold. If a neighborhood includes at least *MinPts* points, it will be considered as a dense region.

Definitions I

- Types of points:
  - Core points
  - Border points
  - Noise points(outliers)

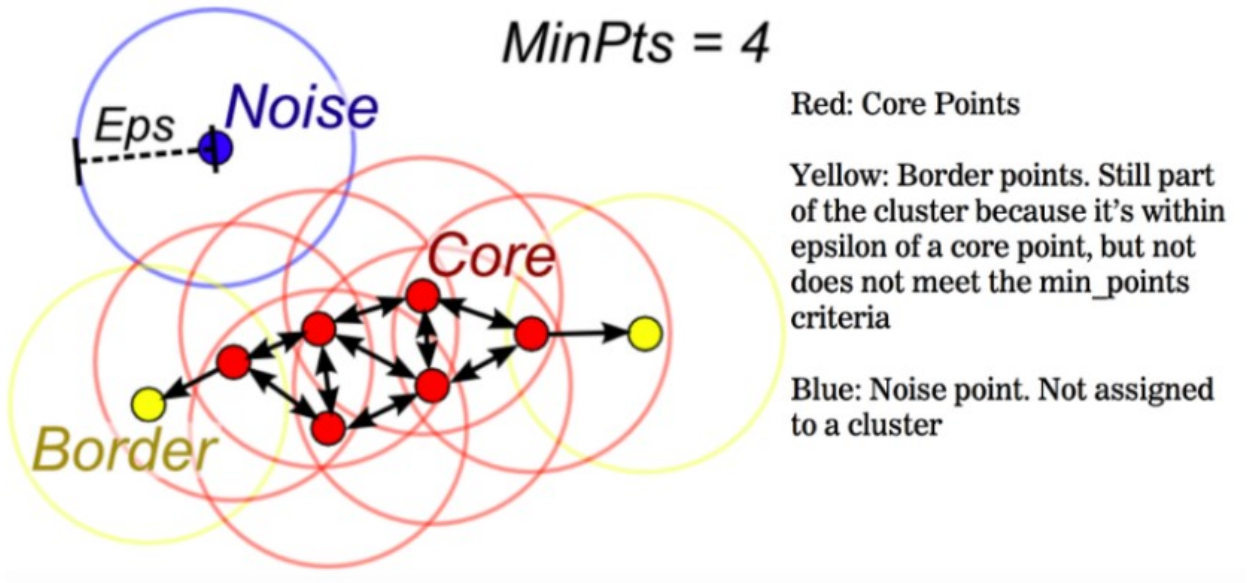


Figure: Different types of points.

Definitions II

- Reachability and Connectivity:
  - **Directly Density-Reachable**: A point X is directly density-reachable from point Y w.r.t *Eps*,*MinPts* if,
    * X belongs to the ϵ neighborhood of Y, i.e, $dist(X, Y) \leq \epsilon.(X \in N_\epsilon(Y))$
    * Y is a core point i.e, $N_\epsilon(Y) \geq MinPts$.

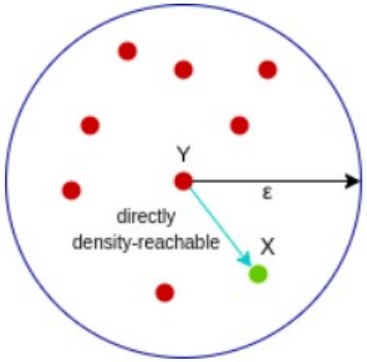


Figure: Example of Directly Density reachable point

Definitions III

- **Density-Reachable**: A point X is density-reachable from point Y w.r.t *Eps*,*MinPts* if there is a chain of points $p_1, \ldots, p_n$, $p_1 = X$, $p_n = Y$ such that p_{i+1} is Directly Density-reachable from p_i i,e, all points on the path are core points, with the possible exception of point $p_n = Y$.

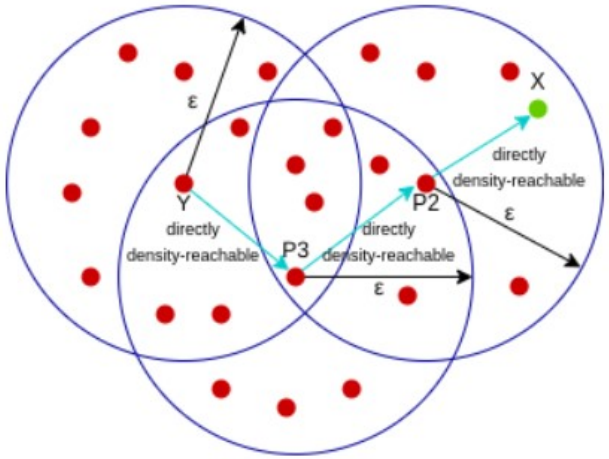


Figure: Example of Density reachable point

Definitions IV

- **Density-Connected**: A point X is density-connected from point Y w.r.t *Eps*,*MinPts* if there exists a point O such that both X and Y are Density-reachable from O w.r.t to *Eps*,*MinPts*. Density-connectivity is a symmetric relation.

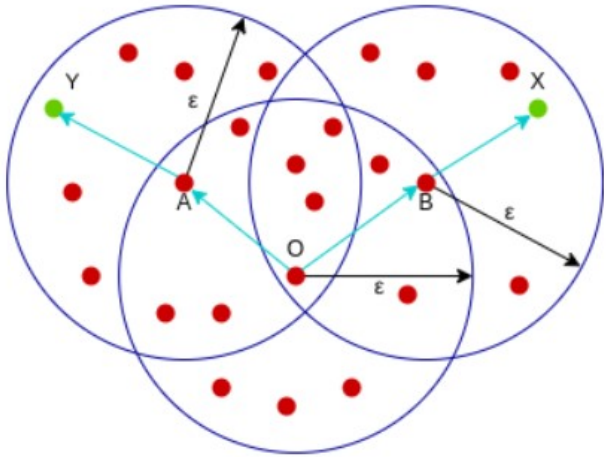


Figure: Example of Density connected points

Parameter selection in DBSCAN I

- DBSCAN is very sensitive to the values of *Eps* and *MinPts*. A slight variation in these values can significantly change the results produced by the DBSCAN algorithm.
- **MinPts**:
  - As a rule of thumb, minimum *MinPts* can be derived from the dimension D of the data set, as $MinPts \geq D + 1$. The low value of $MinPts = 1$ does not make sense, as then every point on its own will already be a cluster.
  - Typically $MinPts = 2 \times dim$ can be used but it may be necessary to choose larger values for very large data, for noisy data.

Parameter selection in DBSCAN II

- **Eps(ϵ)**:
  - If the value of epsilon chosen is too small then a higher number of clusters will be created, and more data points will be taken as noise. Whereas, if chosen too big then various small clusters will merge into a big cluster, and we will lose details.
  - The value for ϵ can then be chosen by using a k-distance graph, plotting the distance to the $k = MinPts - 1$ nearest neighbor ordered from the largest to the smallest value. Good values of ϵ are where this plot shows an ”elbow”.

Parameter selection in DBSCAN III

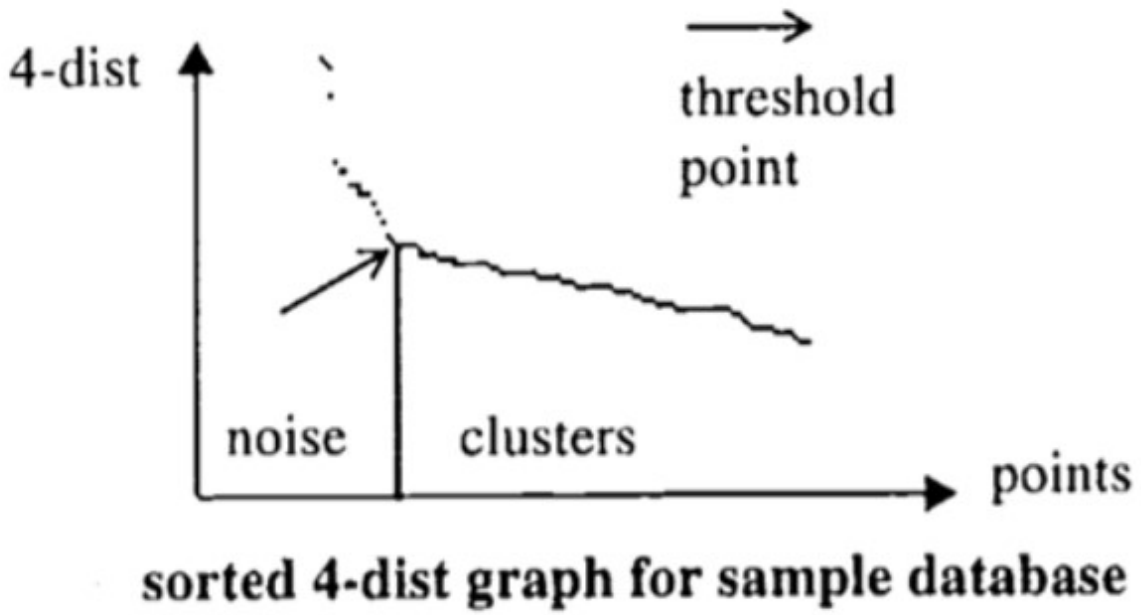


Figure: k-dist graph(for k=4).

DBSCAN Algorithm I

Input: *DB*: Database
Input: ε: Radius
Input: *minPts*: Density threshold
Input: *dist*: Distance function
Data: *label*: Point labels, initially *undefined*

```
1  foreach point p in database DB do                          // Iterate over every point
2      if label(p) ≠ undefined then continue                  // Skip processed points
3      Neighbors N ← RangeQuery(DB, dist, p, ε)               // Find initial neighbors
4      if |N| < minPts then                                   // Non-core points are noise
5          label(p) ← Noise
6          continue
7      c ← next cluster label                                 // Start a new cluster
8      label(p) ← c
9      Seed set S ← N \ {p}                                   // Expand neighborhood
10     foreach q in S do
11         if label(q) = Noise then label(q) ← c
12         if label(q) ≠ undefined then continue
13         Neighbors N ← RangeQuery(DB, dist, q, ε)
14         label(q) ← c
15         if |N| < minPts then continue                      // Core-point check
16         S ← S ∪ N
```

Figure: Pseudocode of DBSCAN Algorithm.

Pros and Cons of DBSCAN

- **Pros**:
  - DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to *k-means*. It can automatically detect the number of clusters based on your input data and parameters.
  - DBSCAN can find arbitrarily-shaped clusters.
  - DBSCAN can handle noise and outliers. All the outliers will be identified and marked without been classified into any cluster. Therefore, DBSCAN can also be used for Anomaly Detection (Outlier Detection).
- **Cons**:
  - DBSCAN algorithm fails in case of varying density clusters.
  - Sensitive to clustering parameters *Eps*(ϵ) and *MinPts*

Results I

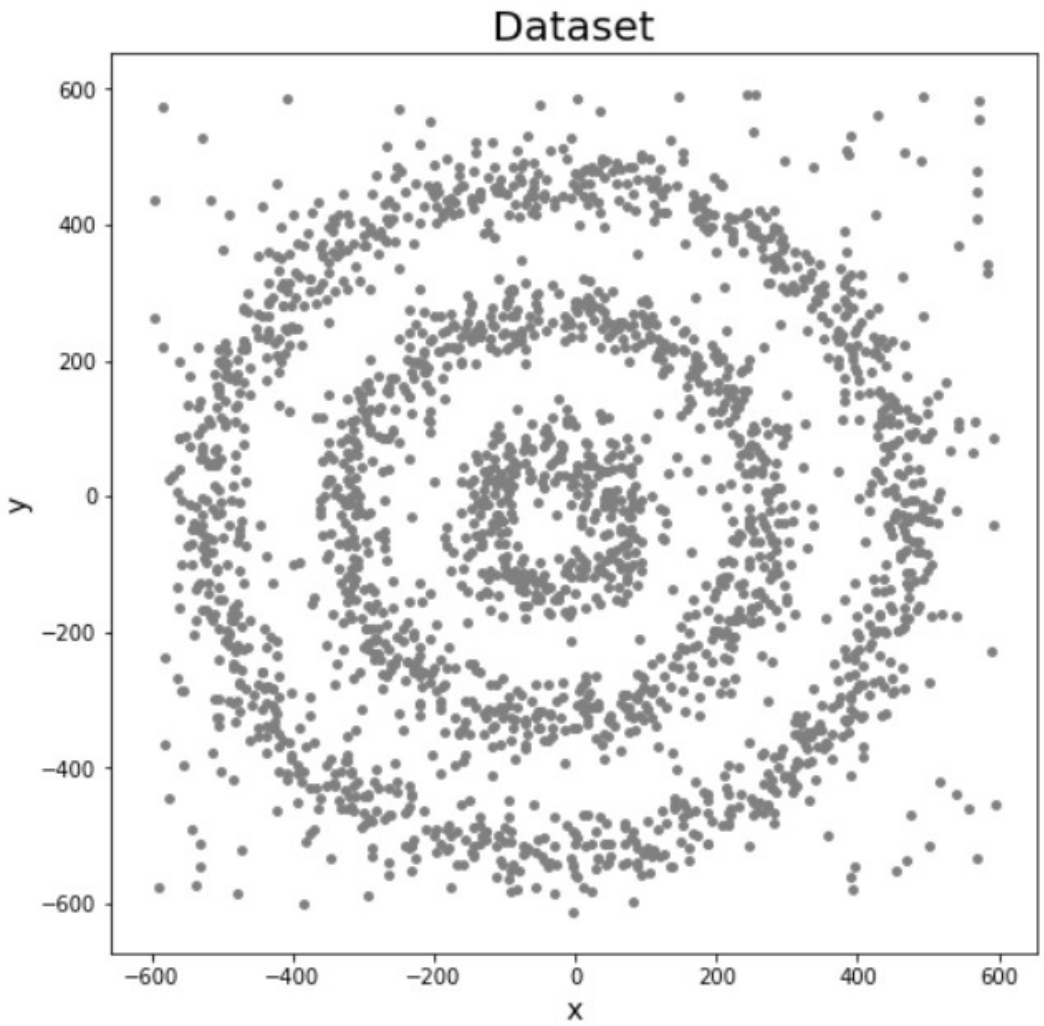
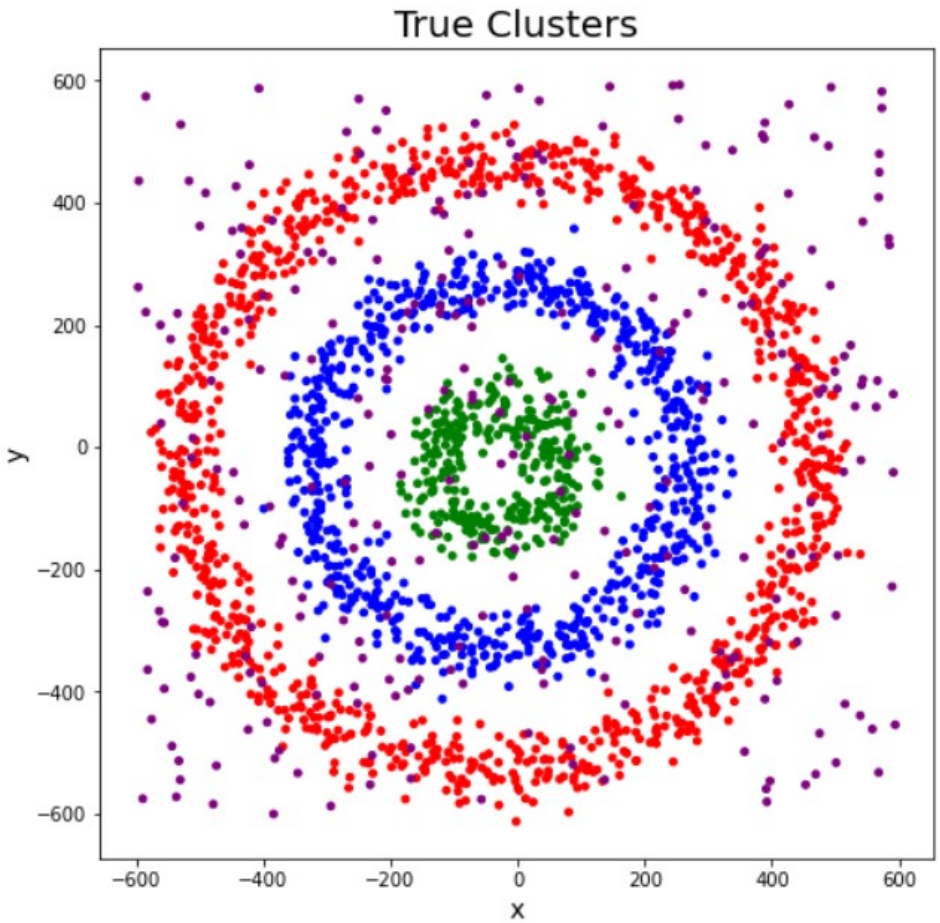


Figure: Dataset.

Results II


True Clusters
600
400
200
0
−200
−400
−600
y
−600
−400
−200
0
200
400
600
x

Results III

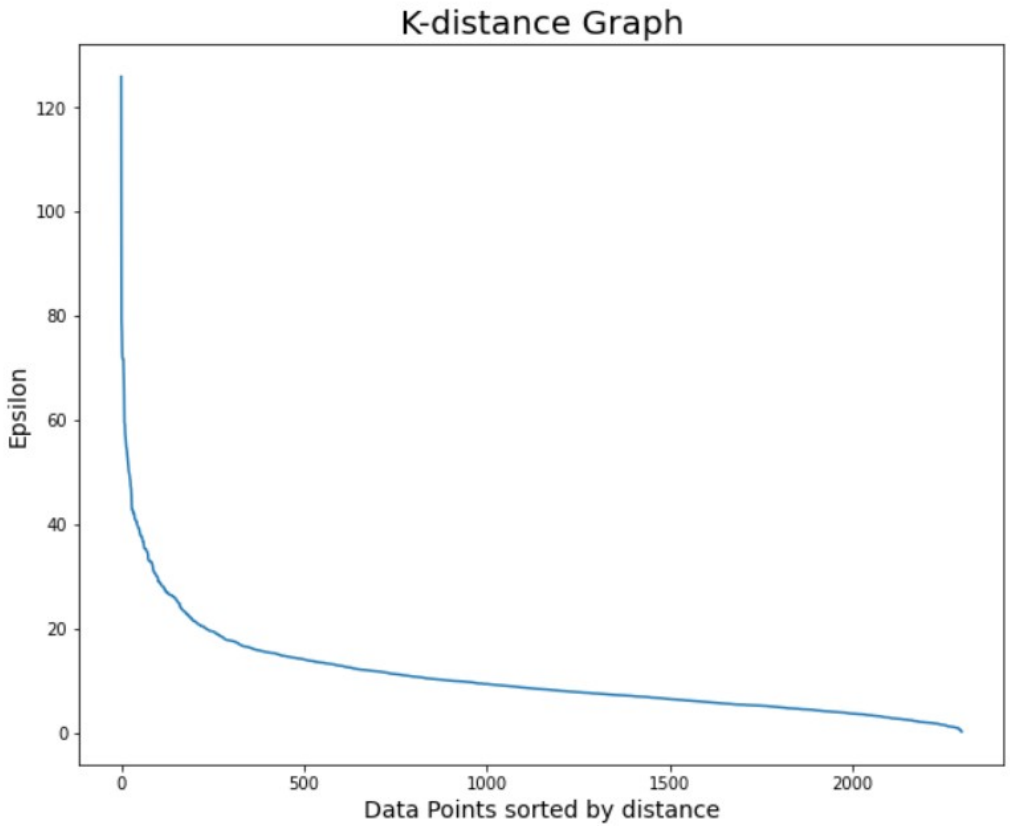


Figure: K-distance graph.

Results IV

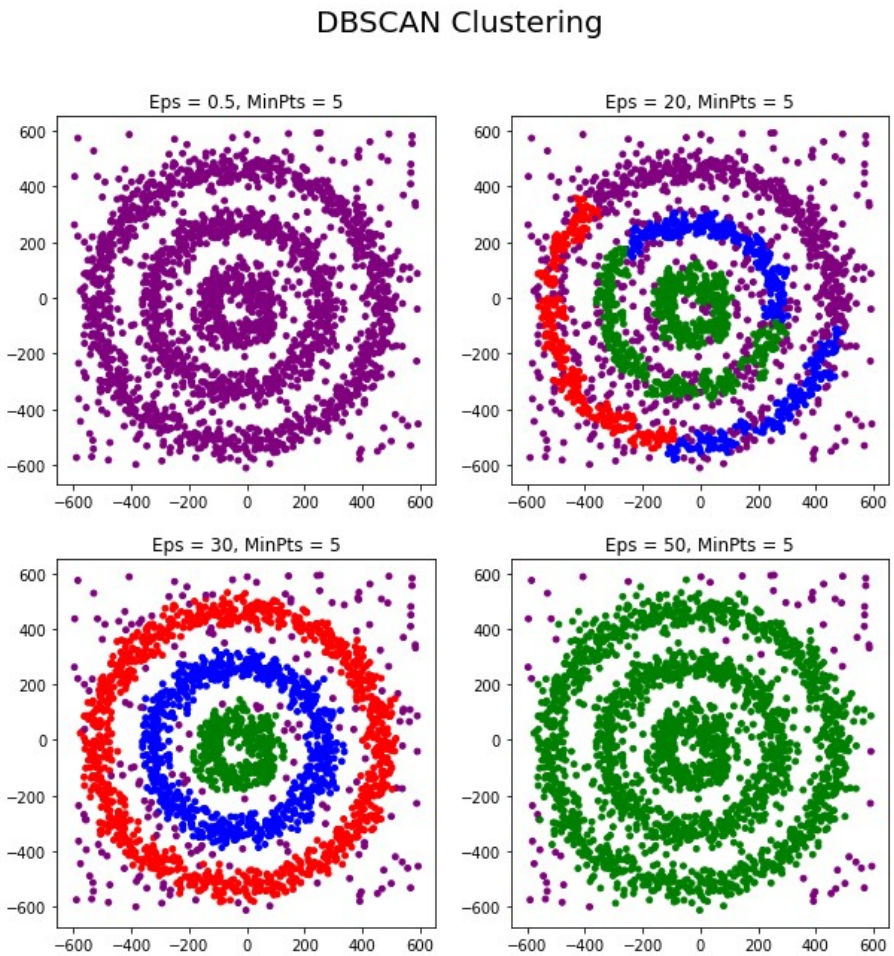


Figure: Clusters for different values of parameters

Spectral clustering

- Spectral clustering is an technique that reduces complex multidimensional data sets into clusters of similar data in rarer dimensions.
- Spectral Clustering uses information from the eigenvalues (spectrum) of special matrices (i.e., Affinity Matrix, Degree Matrix and Laplacian Matrix) derived from the graph or the data set.

Difference between Spectral Clustering and Conventional Clustering Techniques

Spectral clustering is flexible and allows us to cluster non-graphical data as well. It makes no assumptions about the form of the clusters. Clustering techniques, like K-Means, assume that the points assigned to a cluster are spherical about the cluster centre.

Similarity graphs

- Given a set of data points $x_1, \ldots, x_n$ and some notion of similarity $s_{ij} \geq 0$ between all pairs of data points x_i and x_j, the intuitive goal of clustering is to divide the data points into several groups such that points in the same group are similar and points in different groups are dissimilar to each other.
- Each vertex v_i in this graph represents a data point x_i. Two vertices are connected if the similarity s_{ij} between the corresponding data points x_i and x_j is positive or larger than a certain threshold, and the edge is weighted by s_{ij}.

Different similarity graphs

- ϵ-neighborhood graph: Each vertex is connected to vertices falling inside a ball of radius ϵ where ϵ is a real value that has to be tuned in order to catch the local structure of data.
- k-nearest neighbor graph: Here the goal is to connect vertex v_i with vertex v_j if v_j is among the k-nearest neighbors of v_i.

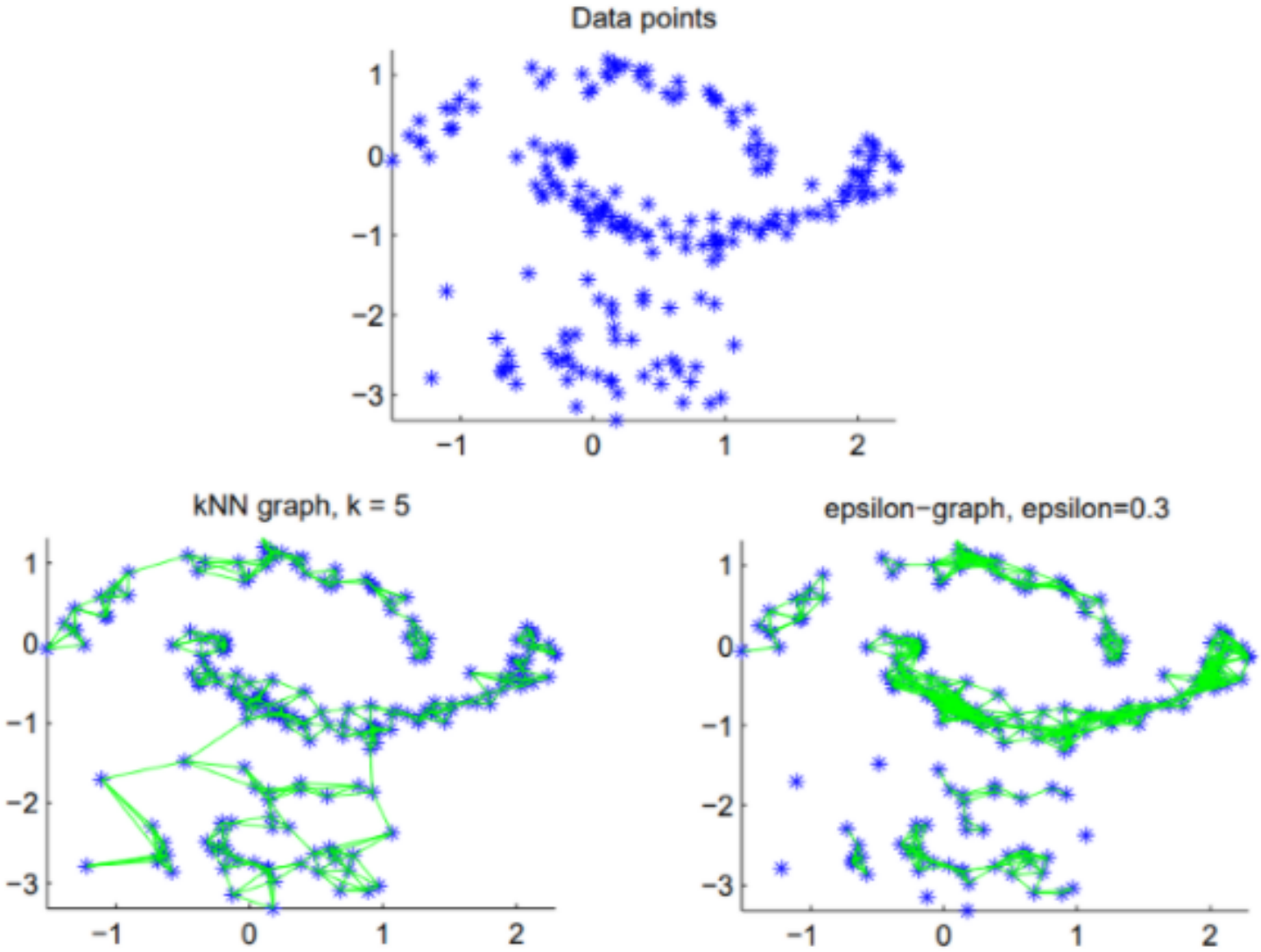


Figure: Different similarity Graph.

- The fully connected graph: Here we simply connect all points with positive similarity with each other, and we weight all edges by s_{ij}. As the graph should represent the local neighborhood relationships, this construction is only useful if the similarity function itself models local neighborhoods. An example for such a similarity function is the Gaussian similarity function $s(x_i, x_j) = \exp(-\|x_i - x_j\|^2/(2\sigma^2))$, where the parameter σ controls the width of the neighborhoods. This parameter plays a similar role as the parameter ϵ in case of the ϵ-neighborhood graph.

Spectral Clustering Matrix Representation

- Adjacency and Affinity Matrix (A): The graph (or set of data points) can be represented as an Adjacency Matrix, where the row and column indices represent the nodes, and the entries represent the absence or presence of an edge between the nodes (i.e., if the entry in row 0 and column 1 is 1, it would indicate that node 0 is connected to node 1).

- Degree Matrix (D): A Degree Matrix is a diagonal matrix, where the degree of a node (i.e., values) of the diagonal is given by the number of edges connected to it. We can also obtain the degree of the nodes by taking the sum of each row in the adjacency matrix.
- Laplacian Matrix (L): This is another representation of the graph/data points, which attributes to the beautiful properties leveraged by Spectral Clustering. One such representation is obtained by subtracting the Adjacency Matrix from the Degree Matrix (i.e., L = D - A).

Properties of Laplacian Matrix

- For every vector $f \in \mathbb{R}^n$, we have

$$f'Lf = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$$

, where w_{ij} is the weight between node i and node j and $w_{ij} \geq 0$.

- L is symmetric and positive semi-definite.
- The smallest eigenvalue of L is 0, the corresponding eigen vector is the constant one vector.
- L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$

Number of connected components and the spectrum of L

Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of L equals the number of connected components $A_1, \ldots, A_k$ in the graph.

- Spectral Gap: The first non-zero eigenvalue is called the Spectral Gap. The Spectral Gap gives us some notion of the density of the graph.
- Fiedler Value: The second eigenvalue is called the Fiedler Value, and the corresponding vector is the Fiedler vector. Each value in the Fiedler vector gives us information as to which side of the decision boundary a particular node belongs to. (Recursive bi-partitioning)
- Using L, we find the first large gap between eigenvalues which generally indicates that the number of eigenvalues before this gap is equal to the number of clusters. (cluster Multiple Eigen vectors)

Spectral Clustering Algorithms

We assume that our data consists of n points $x_1, \ldots, x_n$ which can be arbitrary objects. We measure their pairwise similarities $s_{ij} = s(x_i, x_j)$ by some similarity function which is symmetric and non-negative, and we denote the corresponding similarity matrix by $S = (s_{ij})_{i,j=1,\ldots,n}$. Here we will discuss the algorithm for unnormalized graph laplacian.

Unnormalized spectral clustering

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.

- Construct a similarity graph by one of the ways described in Section 2. Let W be its weighted adjacency matrix.
- Compute the unnormalized Laplacian L.
- **Compute the first k eigenvectors $u_1, \ldots, u_k$ of L.**
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \ldots, u_k$ as columns.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Output: Clusters $A_1, \ldots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

Figure: Unnormalized Spectral Clustering.

Summary

- we first took our graph and built an adjacency matrix.
- We then created the Graph Laplacian by subtracting the adjacency matrix from the degree matrix.
- The eigenvalues of the Laplacian indicated that there were k clusters. The vectors associated with those eigenvalues contain information on how to segment the nodes.
- Finally, we performed k-Means on those vectors in order to get the labels for the nodes. Next, we'll see how to do this for arbitrary data.

Comparison with k-Means

Look at the data below. The points are drawn from two concentric circles with some noise added. We'd like an algorithm to be able to cluster these points into the two circles that generated them.

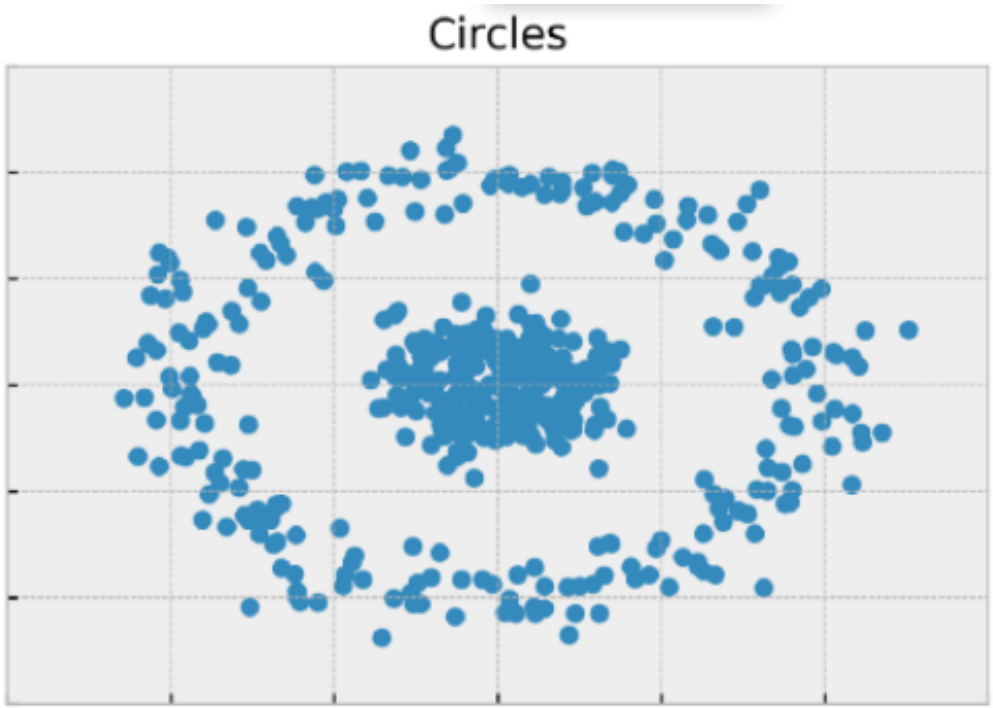


Figure: Circles Data.

This data isn't in the form of a graph. So first, let's just try K-Means. K-Means will find two centroids and label the points based on which centroid they are closest too. Here are the results:

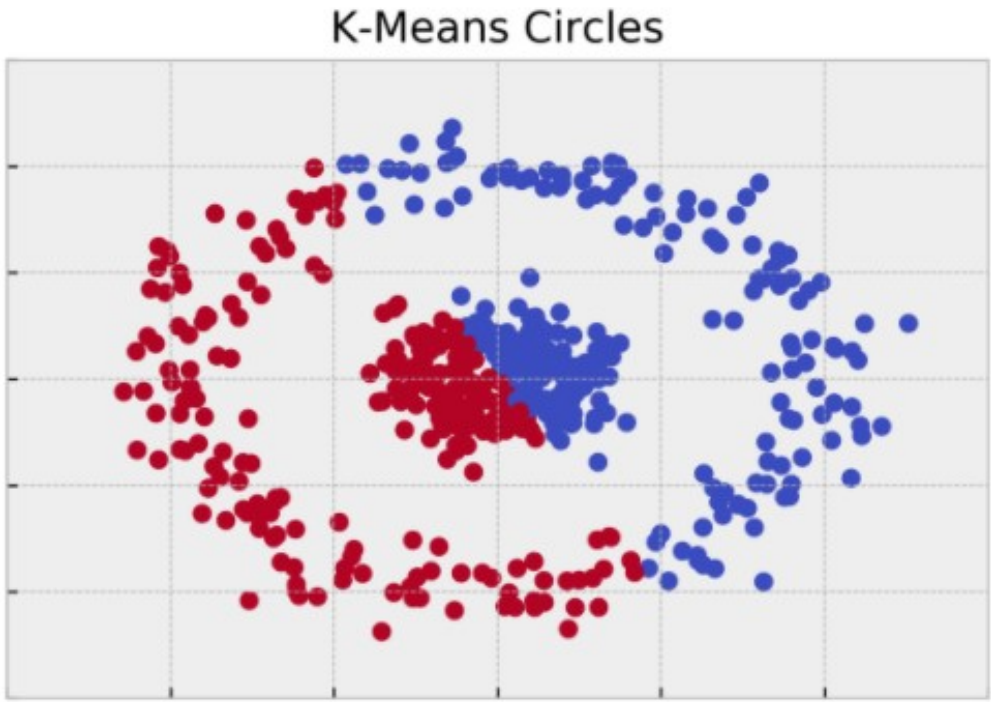


Figure: K-Means Clustering of the Circles Data.

Let's try to tackle this with spectral clustering.

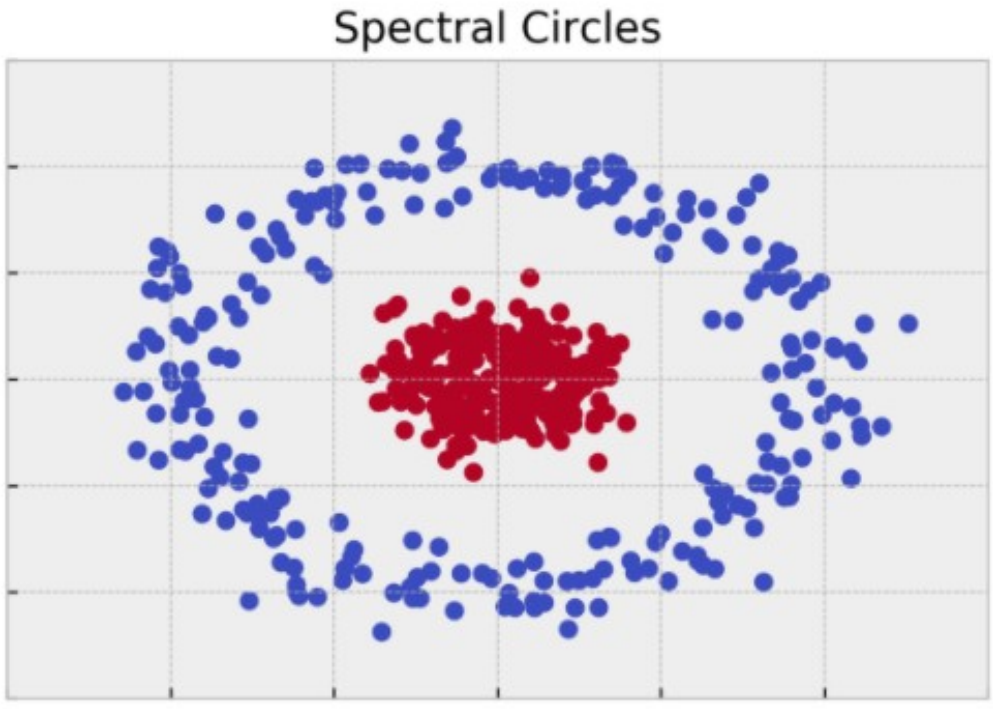


Figure: Spectral Clustering of the Circles Data.

Pros

- Clusters not assumed to be any certain shape/distribution, in contrast to e.g. k-means. This means the spectral clustering algorithm can perform well with a wide variety of shapes of data.
- Do not necessarily need the actual data set, just similarity/distance matrix, or even just Laplacian is enough to perform spectral clustering.

Cons

- Need to choose the number of clusters k, although there is a heuristic to help choosing k value.
- Choice to similarity metric is also a decision variable to choose.Like, to choose a proper kernel like Gaussian kernels, choice of σ is hard.
- Choice of clustering method: k-way vs. recursive bipartite is also a decision to make.