Statistical Pattern Recognition
Project Report

Clustering methods

TEAM MEMBERS

JAYANTH S : 201081003
PRAVEEN KUMAR N : 201082001
RISHABH ROY : 201082002

Contents

List of Figures

Chapter 1

Introduction

- Clustering is a type of unsupervised learning method in machine learning. Clustering 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.
- There are two basic types of clustering algorithms:
  Partitioning and *Hierarchical* algorithms.
- 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. The dendrogram can either be created from bottom-up approach (*agglomerative approach*) or from top-down approach (*divisive approach*) by merging or dividing clusters at each step
- hierarchical algorithms do not need k as an input.However, a termination condition has to be defined indicating when the merge or division process should be terminated.
- Other types of clustering are:
  Density based clustering(Ex: DBSCAN, HDBSCAN, OPTICS).
  Fuzzy clustering(soft clustering)(Ex: *fuzzy c-means*).

Chapter 2

Hierarchical clustering

2.1 Introduction

Hierarchical clustering is a unsupervised learning algorithm in which, a hierarchy of clusters are built. There are two different approaches in Hierarchical clustering.

- Agglomerative Approach
- Divisive Approach

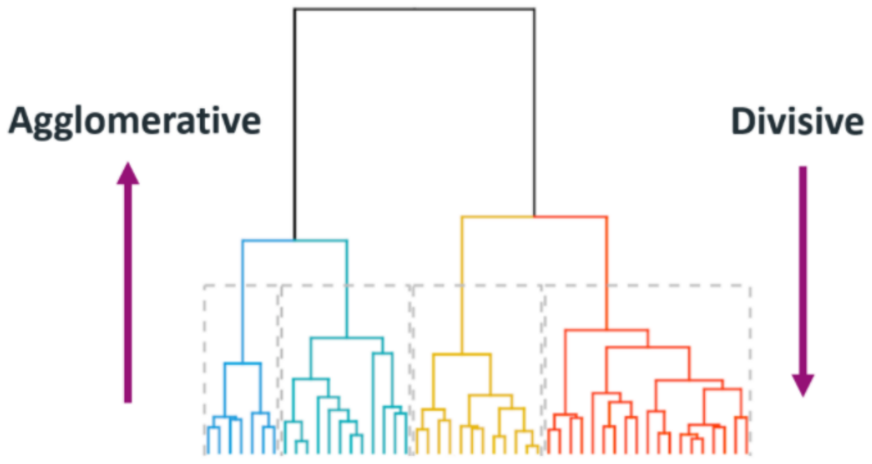


Figure 2.1: Dendrogram that helps in representing the results of hierarchical clustering [1]

2.1.1 Agglomerative Approach

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.

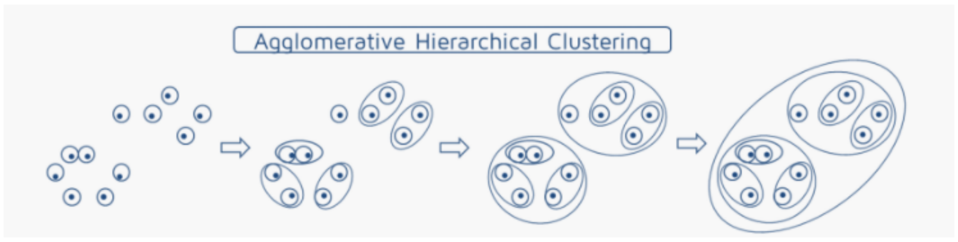


Figure 2.2: [2]

2.1.2 Divisive Approach

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.

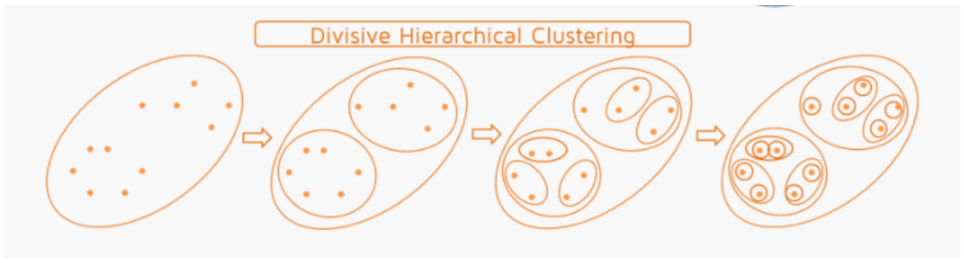


Figure 2.3: [2]

2.2 Basis on which Clusters are Combined or Split

We combine or split the clusters by the use of

- Appropriate metric (Measures the distance between any two points)

- Linkage criterion (measures inter-cluster distances)

2.2.1 Metric

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.
$$\|a-b\|_\infty = \max_i |a_i - b_i|$$

- p-norm ($p > 2$)
$$||x||_p = \left(\sum_i |x_i|^p\right)^{1/p}$$

2.2.2 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.
$$d_{min}(\mathcal{D}_i, \mathcal{D}_j) = \min_{x \in \mathcal{D}_i, x' \in \mathcal{D}_j} \|x - x'\|$$

- 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.
  $$d_{max}(\mathcal{D}_i, \mathcal{D}_j) = \max_{x \in \mathcal{D}_i, x' \in \mathcal{D}_j} \|x - x'\|$$
- Mean-linkage criterion :
  The mean vector for each cluster is found and the distance between them is the distance between the clusters.
  $$d_{mean}(\mathcal{D}_i, \mathcal{D}_j) = \|m_i - m_j\|$$
  where m_i : mean vector for $\mathcal{D}_i$ cluster, m_j : mean vector for $\mathcal{D}_j$ cluster.

2.3 Agglomerative Clustering Process

- 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 ($\bar{k}$) for the given samples, then we can stop when number of clusters is equal to $\bar{k}$.

2.4 Comparison with k-means clustering

- No need to specify number of clusters in hierarchical clustering whereas in k-means we have to specify the number of clusters.

2.5 Cons of Hierarchical Clustering

- Doesn't provide the best solution always. This is because for higher dimension we cann't visualise the data, hence deciding the metric and linkage criterion for clustering becomes important. If they are not chosen properly then we may not get the best solution.
- When the feature vectors of sample data contain different data types, computing the distance between different clusters at each stage is difficult. This is because different data types require different metrics, hence the distance calculation would become complex.

Chapter 3

DBSCAN

3.1 Introduction

- 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,**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise (DBSCAN) was proposed.
- Both *k-means* and *hierarchical* algorithms fail to cluster the dataset with arbitrary cluster shapes, whereas DBSCAN can find arbitrary shape clusters.
- The main concept of DBSCAN algorithm is to locate regions of high density that are separated from one another by regions of low density.
- The key idea is that for each point of a cluster the neighborhood of a given radius has to contain at least a minimum number of points, i.e. the density in the neighborhood has to exceed some threshold.
- The shape of a neighborhood is determined by the choice of a distance function for two points p and q, denoted by $dist(p, q)$.
  For instance, when using the Manhattan distance in $2D$ space, the shape of the neighborhood is rectangular and if we use Eucleidean distance the shape of the neighborhood is circular.

- In the case of DBSCAN, instead of guessing the number of clusters,we will define two hyperparameters: $Eps(\epsilon)$ 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**-neighborhood**: 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. Alternatively, a point will be considered as dense if there are at least $MinPts$ points in its ϵ-neighborhood.

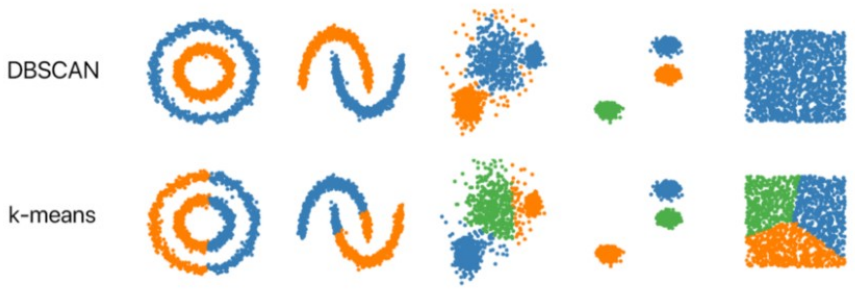


Figure 3.1: DBSCAN vs k-means on various datasets. [3]

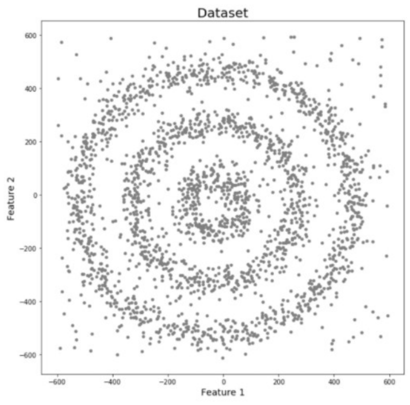


Figure 3.2: Example dataset. [4]

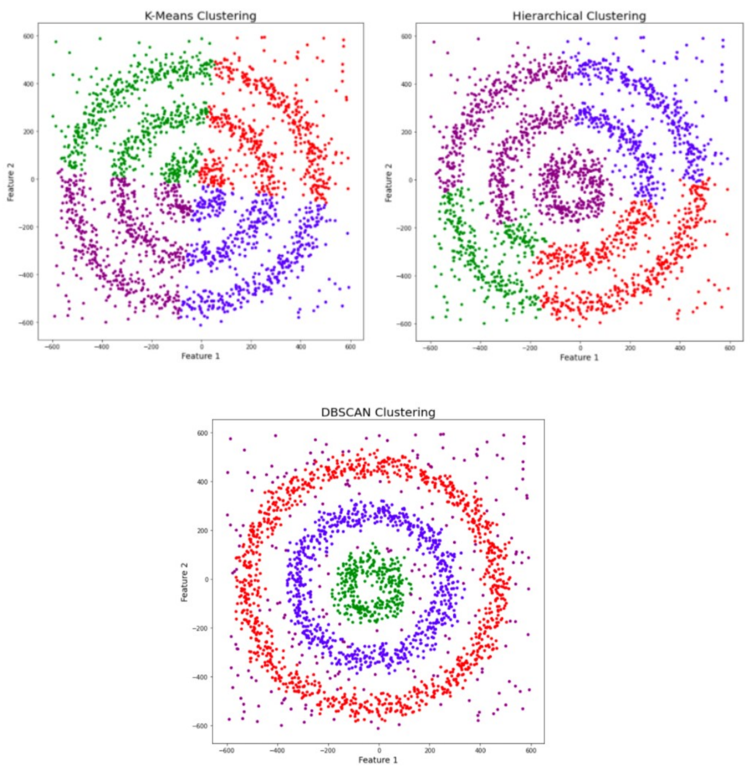


Figure 3.3: Comparision of different clustering methods. Here we want dataset to be clustered into 3 clusters and set of outliers(purple color). [4]

3.2 Definitions

3.2.1 Types of points:

Core points:

A data point p is a Core point if its ϵ-neighborhood contains at least $MinPts$ number of points i.e, if $|N_{\epsilon}(p)| \leq MinPts$. These are points that are at the interior of a cluster.

Border points:

A Border Point has fewer than $MinPts$ within its ϵ-neighborhood, but it lies in the neighborhood of another core point.

Noise points(outliers):

A point that is neither a core point nor a border point is called as noise point. These points are outliers that are not associated with any dense clusters.

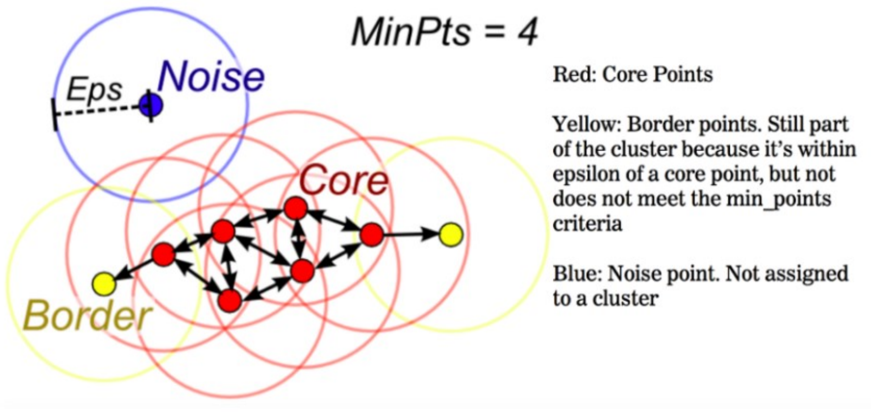

MinPts = 4
Eps
Noise
Core
Border
Red: Core Points
Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria
Blue: Noise point. Not assigned to a cluster

Figure 3.4: Different types of points in DBSCAN. [5]

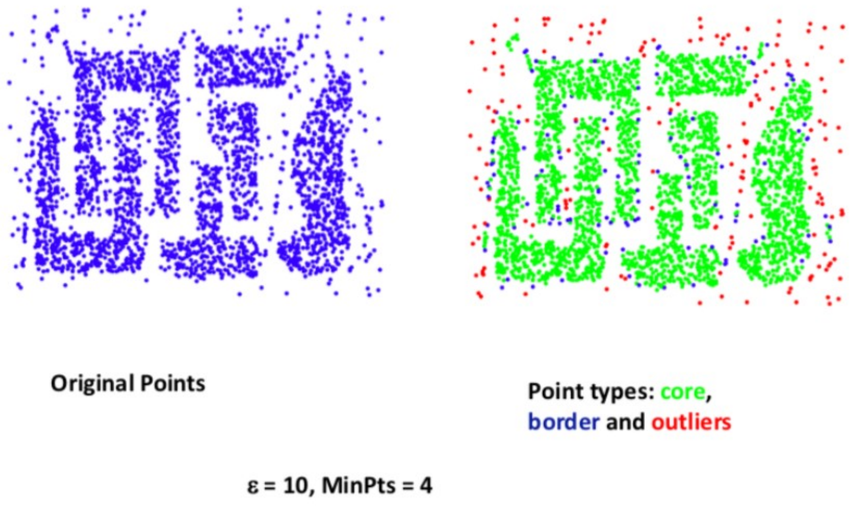


Figure 3.5: Demonstrating dense clusters separated from outliers. [5]

3.2.2 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, $N_\epsilon(Y) \geq MinPts$.

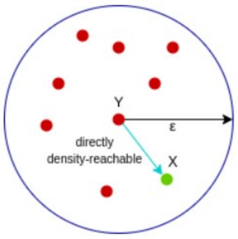


Figure 3.6: Example of Directly Density reachable point. [4]

- Directly Density-reachable is symmetric for pairs of core points. In general, however it is not symmetric if one core point and one border point are involved.

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 $p_n = Y$.

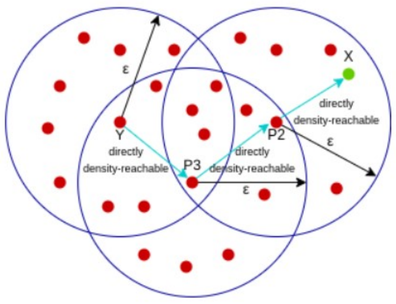


Figure 3.7: Example of Density reachable point. [4]

- Density-reachability is a extension of Direct Density-reachability. This relation is transitive, but it is not symmetric. Although not symmetric in general,Density-reachability is symmetric for core points.
- idea of density-reachable is dependent on value of ϵ. By picking larger values of ϵ, more points become Density-reachable, and by choosing smaller values of ϵ, less points become Density-reachable.

Density-Connected:

- There can be cases when 2 border points will belong to the same cluster C but they don't share a specific core point i.e, not density reachable from each other because the core point condition might not hold for both of them. However, there must be a core point in C from which both border points of C are density-reachable.
- 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.

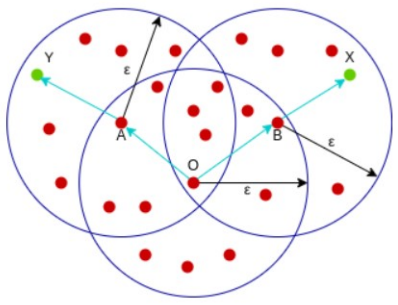


Figure 3.8: Example of Density connected points. [4]

3.3 Parameter Selection in DBSCAN

DBSCAN is very sensitive to the values of *Eps* and *MinPts*. Therefore, it is very important to understand how to select the values of *Eps* and *MinPts*. A slight variation in these values can significantly change the results produced by the DBSCAN algorithm.

3.3.1 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. with $MinPts \leq 2$, the result will be same as of hierarchical clustering with the single link metric, with the dendrogram cut at height ϵ.
- Therefore,$MinPts$ must be chosen at least as 3. As a rule of thumb, $MinPts = 2 \times D$ can be used, but it may be necessary to choose larger values for very large data, for noisy data or for data that contains many duplicates.

3.3.2 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".
- $k - dist$ graphs for $k > 4$ don't differ significantly from the $4 - dist$ graph and they need considerably more computation. Therefore, parameter $MinPts$ can be eliminated by setting it to 4 for all datasets (for 2-dimensional data). The 4-dist value of the threshold point is used as the ϵ value for DBSCAN.

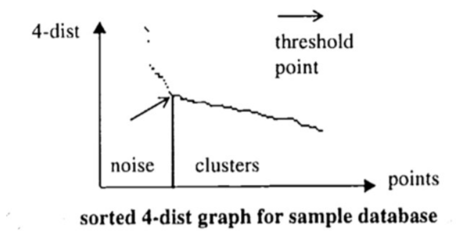
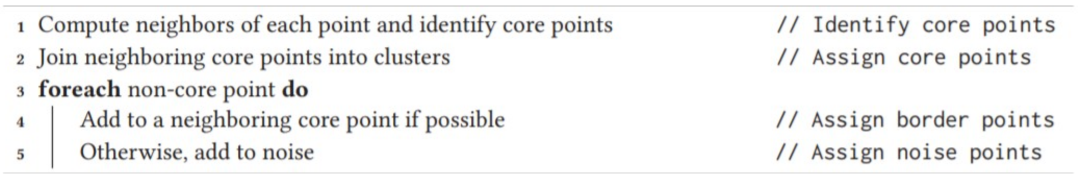


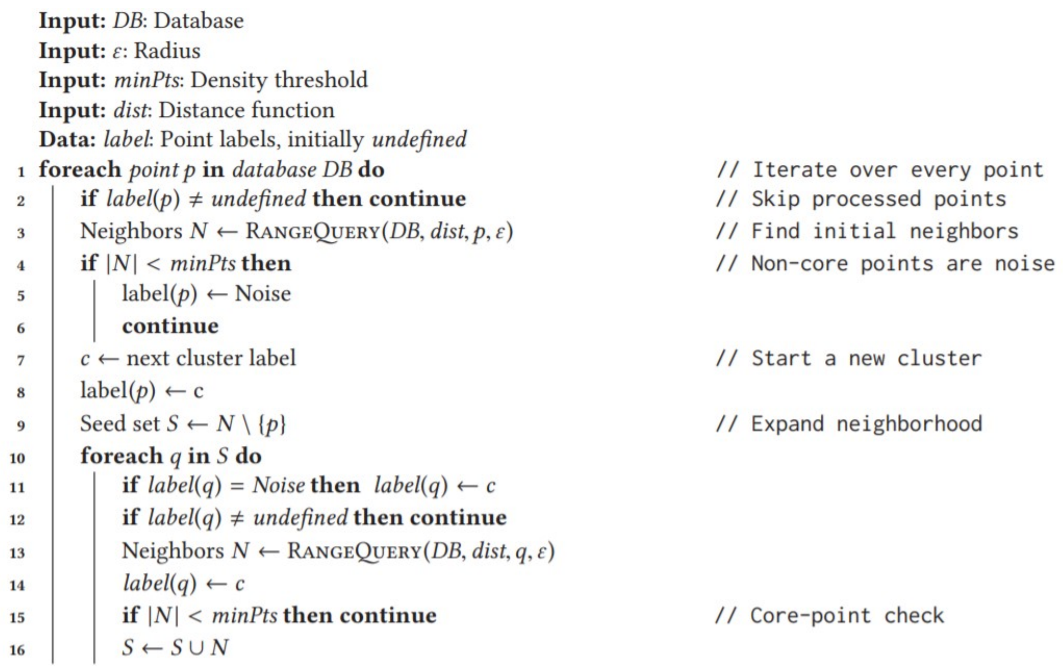
Figure 3.9: k-dist graph(for k=4). [6]

3.4 DBSCAN Algorithm:



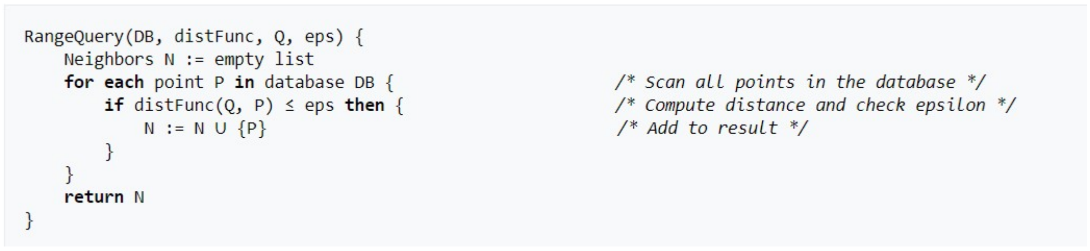
```
Compute neighbors of each point and identify core points   // Identify core points
Join neighboring core points into clusters                 // Assign core points
foreach non-core point do
    Add to a neighboring core point if possible            // Assign border points
    Otherwise, add to noise                                // Assign noise points
```

Figure 3.10: Steps in DBSCAN Algorithm. [7]



```
Input: DB: Database
Input: $\varepsilon$: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
foreach point p in database DB do                          // Iterate over every point
    if $label(p) \neq undefined$ then continue             // Skip processed points
    Neighbors $N \leftarrow$ RangeQuery($DB, dist, p, \varepsilon$)   // Find initial neighbors
    if $|N| < minPts$ then                                 // Non-core points are noise
        label($p$) $\leftarrow$ Noise
        continue
    $c \leftarrow$ next cluster label                      // Start a new cluster
    label($p$) $\leftarrow$ c
    Seed set $S \leftarrow N \setminus \{p\}$              // Expand neighborhood
    foreach $q$ in $S$ do
        if $label(q) = Noise$ then $label(q) \leftarrow c$
        if $label(q) \neq undefined$ then continue
        Neighbors $N \leftarrow$ RangeQuery($DB, dist, q, \varepsilon$)
        $label(q) \leftarrow c$
        if $|N| < minPts$ then continue                    // Core-point check
        $S \leftarrow S \cup N$
```

Figure 3.11: Pseudocode of DBSCAN Algorithm. [7]



```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors N := empty list
    for each point P in database DB {                  /* Scan all points in the database */
        if distFunc(Q, P) ≤ eps then {                 /* Compute distance and check epsilon */
            N := N ∪ {P}                               /* Add to result */
        }
    }
    return N
}
```

Figure 3.12: [8]

3.5 Pros and Cons of DBSCAN:

3.5.1 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).

3.5.2 Cons:

- DBSCAN is not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data are processed.
- DBSCAN algorithm fails in case of varying density clusters.

Chapter 4

Spectral clustering

4.1 Introduction

Spectral clustering is an technique that reduces complex multidimensional data sets into clusters of similar data in rarer dimensions. The main outline is to cluster the all spectrum of unorganized data points into multiple groups based upon their uniqueness.Spectral Clustering uses the connectivity approach to clustering', wherein communities of nodes (i.e., data points) that are connected or immediately next to each other are identified in a graph. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters. 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.

4.2 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. This is a strong assumption and may not always be relevant. In such cases, Spectral Clustering helps create more accurate clusters.

Although, it is computationally expensive for large data sets, since eigen values and eigen vectors need to be computed and clustering is performed on these vectors. Also,for large data sets, the complexity increases and accuracy decreases significantly.

4.3 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. If we do not have more information than similarities between data points, a nice way of representing the data is in form of the similarity graph $G = (V, E)$. 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}.

4.4 Different similarity graphs

There are several popular constructions to transform a given set $x_1, \ldots, x_n$ of data points with pairwise similarities s_{ij} or pairwise distances d_{ij} into a graph. When constructing similarity graphs the goal is to model the local neighborhood relationships between the data points.

4.4.1 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.

4.4.2 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. However, this definition leads to a directed graph, as the neighborhood relationship is not symmetric. There are two ways of making this graph undirected. The first way is to simply ignore the directions of the edges, that is we connect v_i and v_j with an undirected edge if v_i is among the k-nearest neighbors of v_j or if v_j is among the k-nearest neighbors of v_i. The resulting graph is what is usually called the k-nearest neighbor graph. The second choice is to connect vertices v_i and v_j if both v_i is among the k-nearest neighbors of v_j and v_j is among the k-nearest neighbors of v_i. The resulting graph is called the mutual k-nearest neighbor graph.

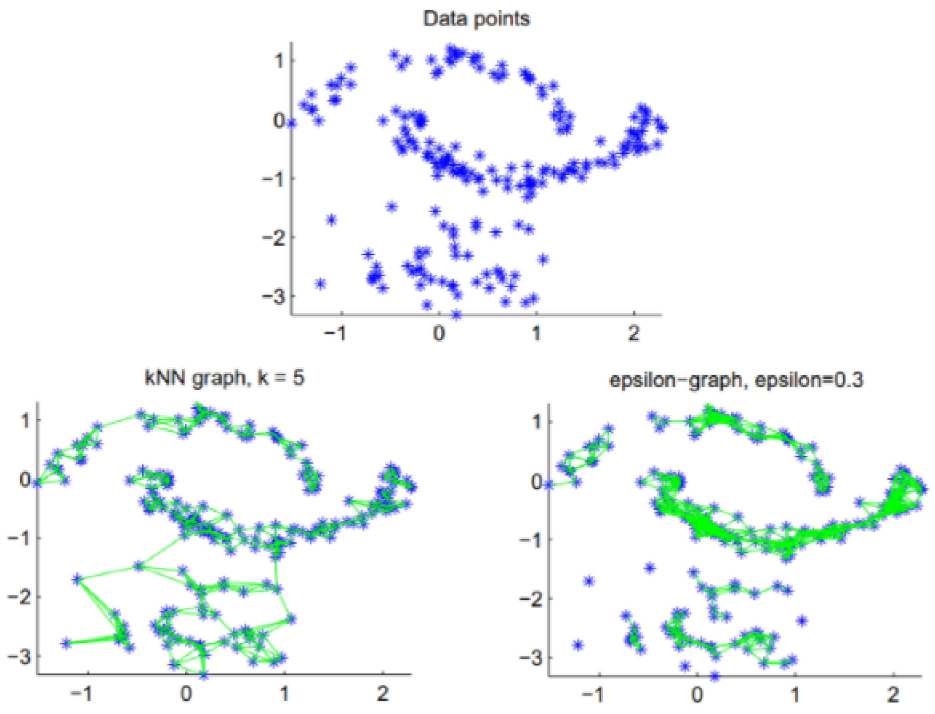


Figure 4.1: Different similarity Graph. [12]

4.4.3 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.

All graphs mentioned above are regularly used in spectral clustering. Theoretical results on the question how the choice of the similarity graph influences the spectral clustering result do not exist.

4.5 Spectral Clustering Matrix Representation

4.5.1 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). An Affinity Matrix is like an Adjacency Matrix, except the value for a pair of points expresses how similar those points are to each other. If pairs of points are very dissimilar then the affinity should be 0. If the points are identical, then the affinity might be 1. In this way, the affinity acts like the weights for the edges on our graph.

4.5.2 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.

4.5.3 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 L:

- For every vector $f \in \mathrm{R}^n$, 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:

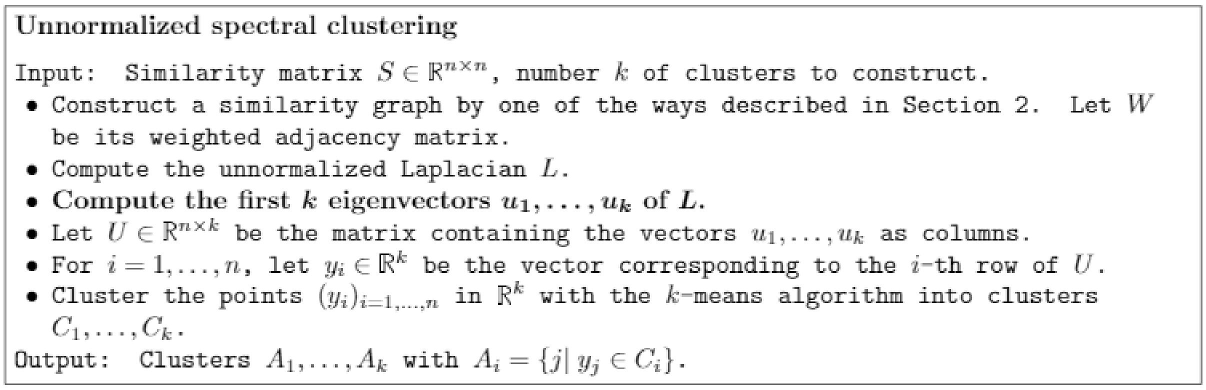
The first non-zero eigenvalue is called the Spectral Gap. The Spectral Gap gives us some notion of the density of the graph.

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. 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.

4.6 Spectral Clustering Algorithms

We assume that our data consists of n points $x_1, \dots, x_n$ 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 $S = (s_{ij})_{i,j=1,\dots,n}$. 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 $u_1, \dots, u_k$ of L.**
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \dots, u_k$ as columns.
- For $i = 1, \dots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- Cluster the points $(y_i)_{i=1,\dots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \dots, C_k$.

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

Figure 4.2: Unnormalized Spectral Clustering. [9]

4.7 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.

4.8 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.

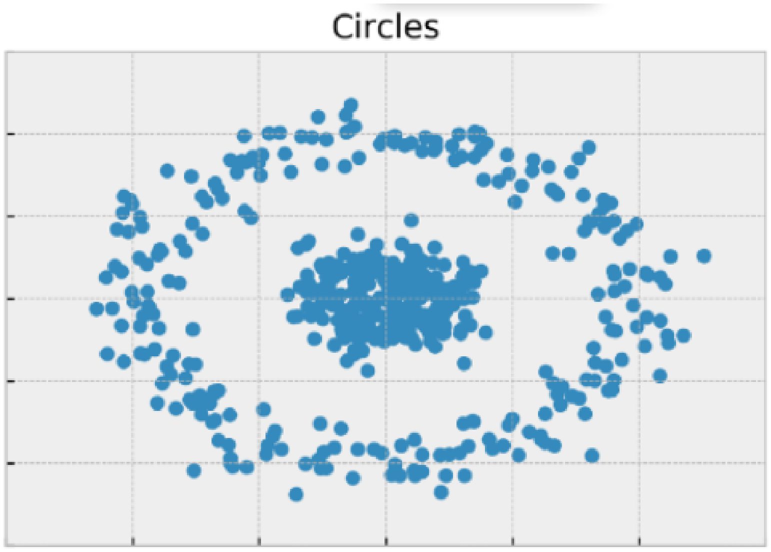


Figure 4.3: Circles Data. [11]

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:

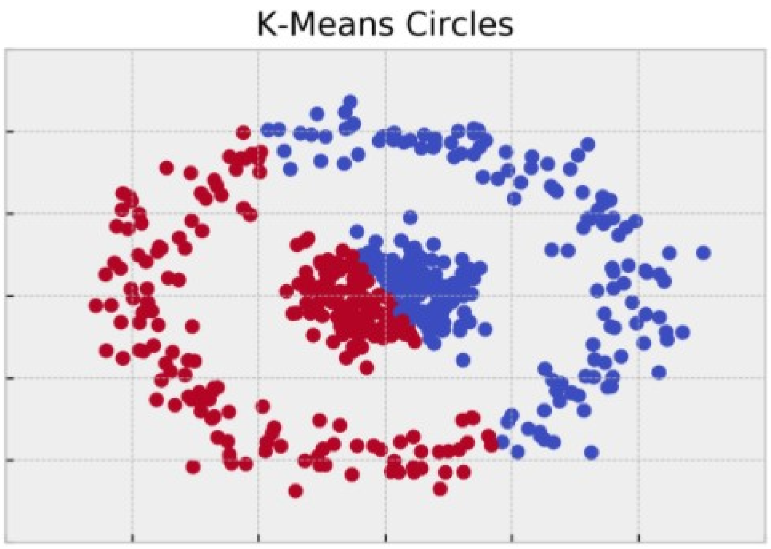


Figure 4.4: K-Means Clustering of the Circles Data. [11]

Obviously, K-Means was not going to work. It operates on euclidean distance, and it assumes that the clusters are roughly spherical. This data (and often real world data) breaks these assumptions. Let's try to tackle this with spectral clustering.

There are a couple of ways to treat our data as a graph. The easiest way is to construct a k-nearest neighbors graph. A k-nearest neighbors graph treats every data point as a node in a graph. An edge is then drawn from each node to its k nearest neighbors in the original space. Generally, the algorithm is not too sensitive of the choice of k. Smaller numbers like 5 or 10 usually work pretty well. Any point in the outer ring should be able to follow a path along the ring, but there won't be any paths into the inner circle. It's pretty easy to see that this graph will have two connected components: the outer ring and the inner circle. Since we're only separating this data into two components, we should be able to use our Fiedler vector trick from before.

Here is the result:

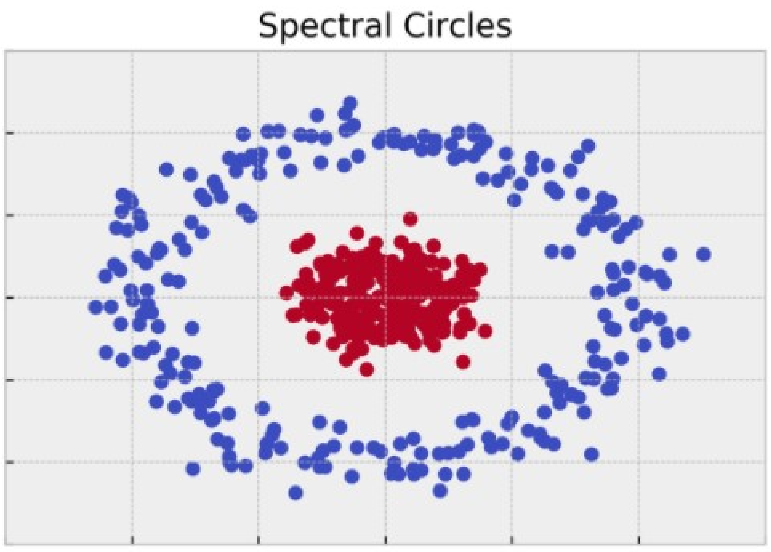


Figure 4.5: Spectral Clustering of the Circles Data. [11]

4.9 Pros and Cons of Spectral clustering

4.9.1 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.

4.9.2 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.

References

[1] https://quantdare.com/hierarchical-clustering/

[2] https://www.mygreatlearning.com/blog/hierarchical-clustering/

[3] https://github.com/NSHipster/DBSCAN

[4] https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/

[5] https://medium.com/@agarwalvibhor84/lets-cluster-data-points-using-dbscan-278c5

[6] https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf?source=post_page

[7] http://www.ccs.neu.edu/home/vip/teach/DMcourse/2_cluster_EM_mixt/notes_slides/revisitofrevisitDBSCAN.pdf

[8] https://en.wikipedia.org/wiki/DBSCAN

[9] https://www.mygreatlearning.com/blog/introduction-to-spectral-clustering/

[10] https://juanitorduz.github.io/spectral_clustering/

[11] https://towardsdatascience.com/spectral-clustering-aba2640c0d5b

[12] https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8

[13] Duda, R. O., Hart, P. E.,, Stork, D. G. (2001). Pattern Classification. New York: Wiley. ISBN: 978-0-471-05669-0