Clustering Methods
(DBSCAN, Spectral, Hierarchical, Optics)

Introduction -

Machine Learning is a broad subject with numerous algorithms and applications in various domains and industries. Unsupervised Learning is one of them, and we can see Clustering in action there.

Unsupervised learning is a method of machine learning that uses unlabeled data. Since we don't know the labels, there is no correct answer for the machine to learn from, but the machine discovers trends in the data and uses them to solve the business problem.

Clustering is a Machine Learning Unsupervised Learning technique that involves the grouping of given unlabeled data. We can cluster the given data points into each group using the Clustering Algorithm in each cleaned data set. The clustering algorithm assumes that the data points that are in the same cluster should have similar properties, while data points in different groups should have highly different properties.

What is the need for Clustering?

Clustering is a widely used ML Algorithm that allows us to find hidden relationships between the data points in our dataset.

Examples -

- Based on a collection of text data, we can organize the data according to the content similarities to create a topic hierarchy.
- Customers are segmented according to similarities of the previous customers and can be used for recommendations.
- In Identifying Fraudulent and Criminal activities.

Various Algorithms -

We have studied four techniques for clustering.

1. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
2. Spectral Clustering
3. Hierarchical Clustering
4. Optics

This report will cover the implementation, pros, and cons of the algorithms' applications, as mentioned earlier.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) -

It is a density-based, commonly used unsupervised clustering algorithm proposed in 1996. Density-based clustering algorithms assume that clusters are dense regions in space separated by regions of lower density. A dense cluster is a region that is "density connected," i.e., the density of points in that region is more significant than a minimum. Since these algorithms expand clusters based on dense connectivity, they can find clusters of arbitrary shapes.

Some important concepts and terms -

First of all, there are two parameters we need to set for DBSCAN, Eps, and MinPts.

Eps: Maximum radius of the neighborhood
MinPts: Minimum number of points in an Eps-neighbourhood of that point

And there is the concept of Directly density-reachable: A point p is directly density reachable from a point q w.r.t. Eps, MinPts, if NEps (q): {p belongs to D | dist(p,q) ≤ Eps} and |N Eps (q)| ≥ MinPts.

Density Reachable (DR): Two points are DR if there is a chain of DDR points that link these two points.

Finally, a point is a core point if it has more than a specified number of points (MinPts) within Eps. These are points that are at the interior of cluster A. And a border point has fewer than MinPts within Eps but is in the neighborhood of a core point. We can also define the outlier(noise) point, which is the points that are neither core nor border points.

How does it work?

1. Randomly select a point p.
2. Retrieve all points density-reachable from p based on Eps and MinPts.
3. If p is a core point, a cluster is formed.

4. If p is a border point, no points are density-reachable from p, and DBSCAN visits the next point of the database.
5. Continue the process until all of the points have been processed.

If a spatial index is used, the computational complexity of DBSCAN is O(N log N), where n is the number of database objects. Otherwise, the complexity is $O(n^2)$.

Case Study -

Our Objective - To categorize the countries using socio-economic and health factors that determine the country's overall development.

Our approach was to apply any clustering algorithm (we used KMeans) and categorize countries based on the clusters we get.

Problem with our approach - If a point is very far from all the clusters, then these algorithms, typically, try to find the best group for that data point. But this might be the case that the data point is an outlier (different from all other countries).

To address the above problem, we tried using DBSCAN and found out that the dataset has lots of noise. There are some countries which are not similar to others.

We observed some pros and cons of DBSCAN.

Pros -

1. There is no need to set the number of clusters.
2. It defines outliers as noise.
3. It helps to find the arbitrarily sized and arbitrarily shaped clusters quite well.
4. It is possible to parallelize DBSCAN. HPDBSCAN algorithm is an efficient parallel version of the DBSCAN algorithm that adopts the core idea of the grid-based clustering algorithm.

Cons -

1. DBSCAN does not perform well on varying density clusters.
2. It also does not perform well with high-dimensional data. We tried DBSCAN on the full dataset (including all features), then it either labels all countries the same or all different (noise).

3. It is challenging to set optimal eps and minpoints because there is no simple way of doing that. We tried to analyze them by 2D plots. But it is not possible to analyze the data in higher dimensions.

Spectral Clustering Algorithm

In recent years, spectral clustering has become one of the most popular modern clustering algorithms. It is simple to implement, can be solved efficiently by standard linear algebra software, and very often outperforms traditional clustering algorithms such as the k-means algorithm. On the first glance spectral clustering appears slightly mysterious, and it is not obvious to see why it works at all and what it really does.

What is Spectral Clustering Algorithm

Spectral clustering is an EDA technique that reduces complex multidimensional datasets 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. It treats data points as the nodes of the graph. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters. It 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.

Each vertex vi in the graph represents a data point Xi. Two vertices are connected if the similarity (sij) between the data points xi and xj is larger than a certain threshold, and the edge is weighted by sij. This problem can be formulated using similarity graph: find a partition of the graph such that the edges between different groups have low weights and the edges within group have high weights

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. The data points in Spectral Clustering should be connected, but may not necessarily have convex boundaries, as opposed to the conventional techniques. It is

computationally expensive for large datasets, since eigenvalues and eigenvectors need to be computed and clustering is performed on these vectors.

Some terms and concepts

Adjacency and Affinity Matrix (A): The graph is 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

Degree Matrix: It is a diagonal matrix, where the value of the diagonal is given by the number of edges connected to it

Laplacian Matrix (L): Representation of the graph/data points, which attributes to the properties leveraged by Spectral Clustering. One such representation is obtained by subtracting the Adjacency Matrix from the Degree Matrix (i.e. L = D – A).

Spectral Gap: The first non-zero eigenvalue is called the Spectral Gap. It 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.

Pseudocode of Spectral Algorithm

1. Construct a similarity graph
2. Determine the Adjacency matrix W, Degree matrix D and the Laplacian matrix L
3. Compute the eigenvectors of the matrix L
4. Using the second smallest eigenvector as input, train a k-means model and use it to classify the data

Case Study

1. Implementing Spectral Algorithm from Scratch using pseudocode

Modelling using different graph and find the best filt

1. Taking dense and connected graph
   Usually it results in classifying the points with less cluster. Also it heavily depends on the dataset

2. Taking sparse graph
   Usually it results in classifying points with more clusters. But it also depends on the dataset

3. Using K means
   This is better than the above two. Using the graph from K-means, determine the number of clusters.

2. Comparing the Spectral Algorithm from scratch (using the pseudocode mentioned above) and sklearn inbuilt spectral algorithm

It is observed that the sklearn and implemented algorithm give the similar graph

3. Implementing a case study on the dataset from Kaggle (credit Card dataset for clustering)

In this case we try to develop a customer segmentation to define marketing strategy. The sample Dataset summarizes the usage behavior of about 9000 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioral variables.

The 18 behaviour include:

1. CUST*ID* : Identification of Credit Card holder (Categorical)
2. BALANCE : Balance amount left in their account to make purchases (
3. BALANCE FREQUENCY : How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
4. PURCHASES : Amount of purchases made from account
5. ONE OFF *PURCHASES* : Maximum purchase amount done in one-go
6. INSTALLMENTS PURCHASES : Amount of purchase done in installment
7. CASH*ADVANCE* : Cash in advance given by the user
8. PURCHASES FREQUENCY : How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
9. ONE OFF PURCHASES FREQUENCY : How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
10. PURCHASES INSTALLMENTS FREQUENCY : How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
11. CASHADVANCE FREQUENCY : How frequently the cash in advance being paid

12. CASH ADVANCE : Number of Transactions made with "Cash in Advanced"
13. PURCHASES_Trx : Number of purchase transactions made
14. CREDIT LIMIT : Limit of Credit Card for user
15. PAYMENTS : Amount of Payment done by user
16. MINIMUM_PAYMENTS : Minimum amount of payments made by user
17. PRCFULL PAYMENT : Percent of full payment paid by user
18. TENURE : Tenure of credit card service for user

Steps:-

1. Importing the required libraries
2. Loading and Cleaning the Data
3. Preprocessing the data to make the data visualizable (which helps in understanding the dataset)
4. Building the Clustering models and Visualizing the clustering
5. Evaluating and comparing the performance (e.g. from K-means)

Observation

Personally to me this seems like better clustering as it really segments out the top half of customers having more than usual amount of usage of credit card and the customers which have very low usage. This seems to be a more actionable result if we want to direct our marketing strategies according to usage of credit card.

Hierarchical clustering Algorithm

This algorithm groups the similar object called clusters. The approach in this algorithm is Hierarchical.
Types of Hierarchical clustering algorithm-
i) Agglomerative
ii) Divise

Agglomerative clustering

The Agglomerative clustering is used to group the objects in clusters based on their similarity. It is also known as AGNES(Agglomerative Nesting). The algorithm starts by treating each object as a singleton cluster that has been merged into one big cluster containing all objects. The result is a tree-based representation of the objects called DENDROGRAM.

Pros and cons:
No assumption of particular number of clusters
It is slow for large sets.
Time complexity is O(n^2logn).

How it works:-

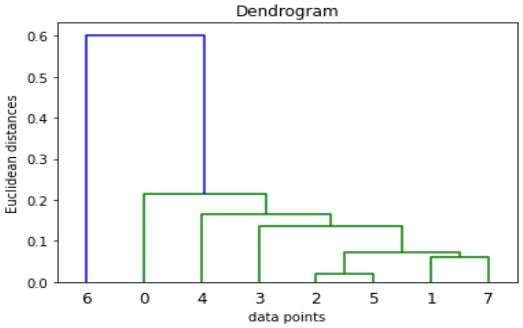
step-1) Make each data point as a single cluster, so initially there are n(no. of data points) clusters.

step-2) Then, take two closest clusters and make them one cluster.
step-3) Repeat step 2 until there is only one cluster.

Dendrogram-Tree like structure

To visualize the history of grouping and figure out the optimal number of clusters.

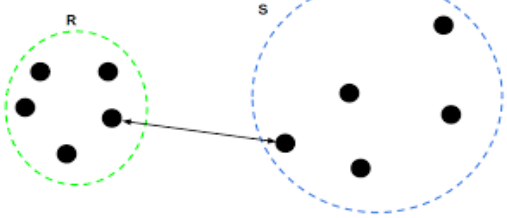
1) Determine the largest distance that doesn't intersect any of the other clusters.
2) Draw a horizontal line at both extremities.
3) The Optimal number of clusters is equal to the number of vertical lines going through the horizontal line.



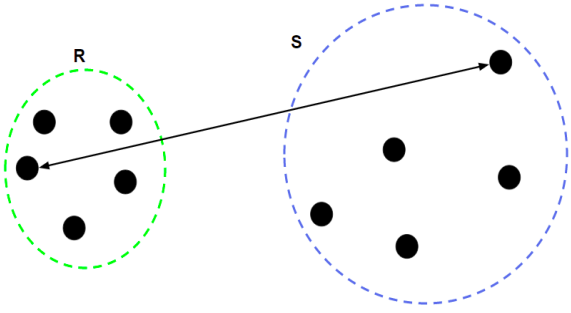
How to find out which clusters are closest?

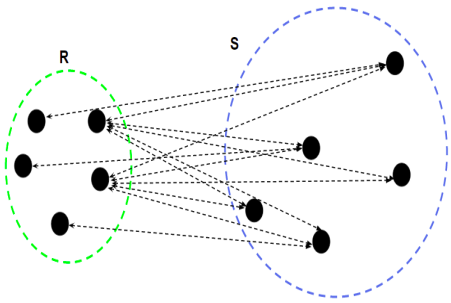
There are several methods to find out the which two clusters are closest

Single linkage

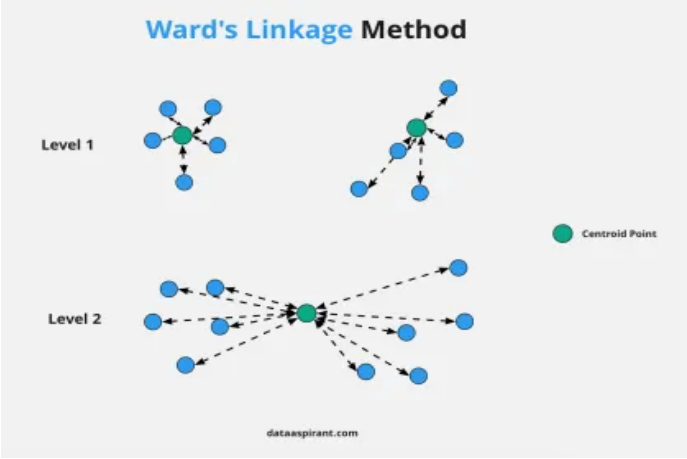


2) complete Linkage





3) Average linkage



4) ward Linkage

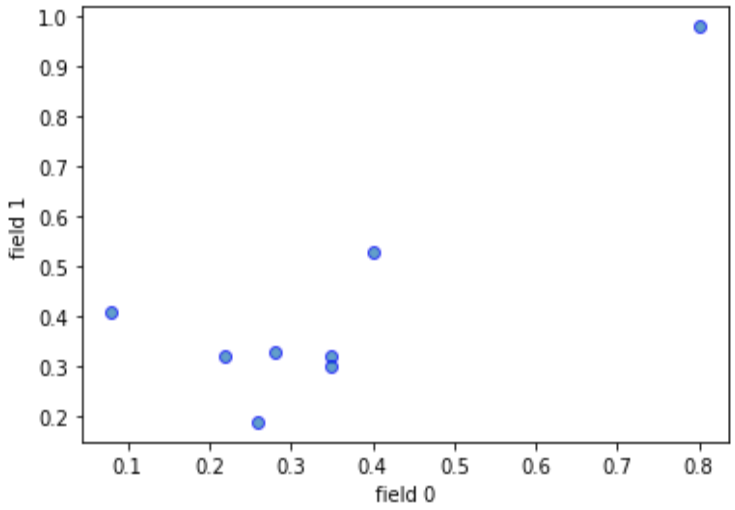
Example:

There will be a matrix that will contain the distance between each pair of clusters at every iteration. As we are reducing the number of clusters by one at each iteration, so the size of the matrix will also be reduced by 1 that means the number of rows will reduce by 1 and the number of columns will also be reduced by 1.

Let the dataset is:

1 [0.40,0.53],
2 [0.22,0.32],
3 [0.35,0.32],
4 [0.26,0.19],
5 [0.08,0.41],
6 [0.35,0.30],
7 [0.80,0.98],
8 [0.28,0.33]

The representation of this dataset on graph is given below:



Now, On applying the agglomerative clustering algorithm, we will find out two clusters which need to be merged from that distance matrix.

Initially:-

```
Sample size before clustering  :  8
Cluster  1(going to be merged) : [2]
Cluster  2(going to be merged) : [5]
Current Sample                 : [[0], [1], [[2, [5]]], [3], [4],
[6], [7]]
Cluster we get                   : [[2, [5]]]
Sample size after clustering     : 7
```

Iteration -1)

```
Sample size before clustering    :  7
Cluster  1(going to be merged)   : [1]
Cluster  2(going to be merged)   : [7]
Current Sample                   : [[0], [[1, [7]]], [[2, [5]]],
[3], [4], [6]]
Cluster we get                   : [[1, [7]]]
Sample size after clustering     : 6
```

Iteration -2)

```
Sample size before clustering    :  6
Cluster  1(going to be merged)   : [[1, [7]]]
Cluster  2(going to be merged)   : [[2, [5]]]
Current Sample                   : [[0], [[[1, [7]], [[2, [5]]]]],
[3], [4], [6]]
Cluster we get                   : [[[1, [7]], [[2, [5]]]]]
Sample size after clustering     : 5
```

Iteration -3)

```
Sample size before clustering    :  5
Cluster  1(going to be merged)   : [[[1, [7]], [[2, [5]]]]]
Cluster  2(going to be merged)   : [3]
Current Sample                   : [[0], [[[[1, [7]], [[2, [5]]]],
[3]]], [4], [6]]
Cluster we get                   : [[[[1, [7]], [[2, [5]]]], [3]]]
Sample size after clustering     : 4
```

Iteration -4)

```
Sample size before clustering    :  4
Cluster  1(going to be merged)   : [[[[1, [7]], [[2, [5]]]], [3]]]
Cluster  2(going to be merged)   : [4]
Current Sample                   : [[0], [[[[[1, [7]], [[2, [5]]]],
[3]], [4]]], [6]]
Cluster we get                   : [[[[[1, [7]], [[2, [5]]]], [3]],
[4]]]
Sample size after clustering     : 3
```

Iteration -5)

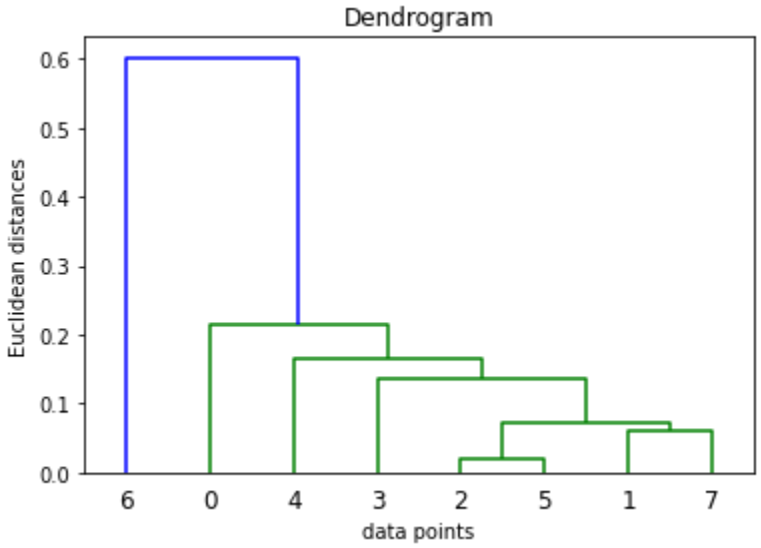
```
Sample size before clustering    :  3
Cluster  1(going to be merged)   : [0]
Cluster  2(going to be merged)   : [[[[[1, [7]], [[2, [5]]]], [3]],
[4]]]
Current Sample                   : [[[0, [[[[[1, [7]], [[2, [5]]]],
[3]], [4]]]]], [6]]
Cluster we get                   : [[0, [[[[[1, [7]], [[2, [5]]]],
[3]], [4]]]]]
Sample size after clustering     : 2
```

Iteration -6)

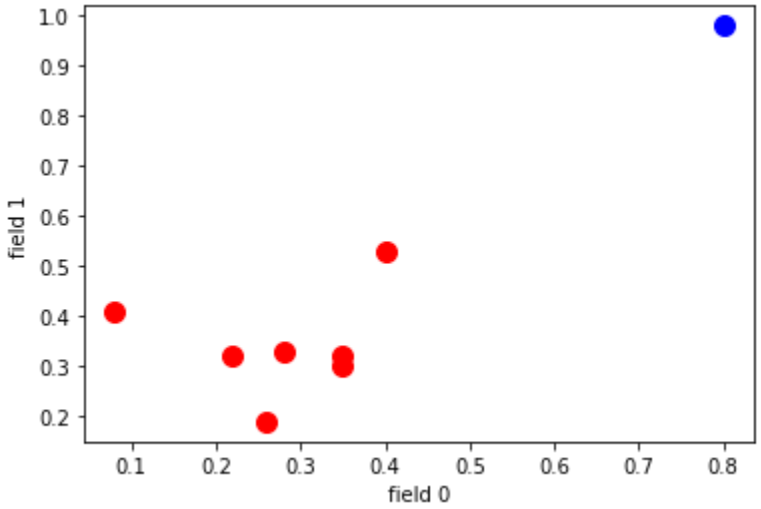
```
Sample size before clustering    :  2
Cluster  1(going to be merged)   : [[0, [[[[[1, [7]], [[2, [5]]]],
[3]], [4]]]]]
Cluster  2(going to be merged)   : [6]
Current Sample                   : [[[[0, [[[[[1, [7]], [[2, [5]]]],
[3]], [4]]]], [6]]]]
Cluster we get                   : [[[0, [[[[[1, [7]], [[2, [5]]]],
[3]], [4]]]], [6]]]
Sample size after clustering     : 1
```

As the number of clusters at this point is 1. Now, we will stop the algorithm.
Now, There could be two ways, we may want the particular number of clusters or we may want the optimal number of clusters. Now, to get the particular number of clusters we will dendrogram as defined above.

Dendrogram of our dataset:-



Now, by observing this dendrogram, we can say the optimal number of clusters are 2.



OPTICS clustering algorithm

Introduction :

- **Ordering points to identify the clustering structure** (OPTICS) is an algorithm for density based clustering.
- It's quite an old algorithm already, as it was presented in 1999.
- It is similar to the DBSCAN for clustering, an extension even, and hence borrows some of its

components as well as its algorithmic components.

- It is also known as relative of DBSCAN.
- It is preferred over DBSCAN when we have clusters of varying densities.

OPTICS components

Epsilon parameter (or ϵ)

It is a distance parameter in the sense that for any point p, the epsilon defines a distance around the point, like this:

MinPts parameter

It is used together with epsilon because it illustrates how many points must be within the ϵ distance of a point p (including the point) in order to form a cluster

Core points

When the point p has minPts within its ϵ distance including itself, we say that it is a core point and that it has sufficient amount of points in its ϵ-neighborhood for becoming one. A core point always represents a cluster. Possibly, it is still in formation, meaning that it will merge with other clusters later.

Core Distance

Core distance is defined as follows. For any point p with some epsilon ϵ and hence an epsilon neighborhood Nϵ(p):

$$\text{core-dist}_{\epsilon,\text{minPts}}(p) = \begin{cases} \text{undefined}, & \text{if } |N_\epsilon(p)| < \text{minPts} \\ \text{minPts-th smallest distance in } N_\epsilon(p), & \text{otherwise} \end{cases}$$

Reachability distance

While the core distance expresses the minimum distance to keep a point a core point, the reachability distance expresses the distance which is reachable from a core point.

It is expressed as follows in terms of an arbitrary point o that is reached from a point p:

$$\text{reach-dist}_{\epsilon,\text{minPts}}(o,p) = \begin{cases} \text{undefined}, & \text{if } |N_\epsilon(p)| < \text{minPts} \\ \max(\text{core-dist}_{\epsilon,\text{minPts}}(p), dist(p,o)), & \text{otherwise} \end{cases}$$

- Points that are part of local clusters can be identified by means of core points, using the concepts of epsilon and minimum number of points borrowed from the DBSCAN algorithm.
- After a local cluster has been identified, points in the vicinity must be identified for whether they are part of the cluster or not. For this reason we compute the core distance, the minimum distance from a point in order to remain a core point, and the reachability distance for another point, which expresses how far away the point is located from that point.

Pseudocode for OPTICS algorithm

- The function OPTICS can be called with a database (DB), and values for epsilon and minimum amount of points.
- For each point in the database, we first set reachability distance to undefined; we must compute it later.
- Then, for each unprocessed point, we perform the following:
    - We get the ϵ-neighborhood for the point. We mark p as processed (we looked at it).
    - We push p to the ordered list (it's the first point we're looking at).
    - We now look at the core distance of p: if it's not undefined (i.e. if it is a core point), we will look further. If it is not a core point, we move on to the next unprocessed point.
        - For core points, we initialize an empty priority queue i.e. a queue where the most important values are read from first. We then call the update function which orders the priority queue based on reachability distance.
        - For the ordered priority queue (where we shall see that lowest reachability distance from the core point p
        - and hence the closest points are covered first), for each point, we get its neighbors. We then mark the point q
        - as processed and output it to the ordered list. If it's a core point as well, we can extend the priority queue as the clusters are close to each other and likely belong to the same bigger cluster. Extending the priority queue through update means that more points are added to the reachability-distance ordered Seeds list.
    - In other words, the algorithm keeps expanding on a particular point *until* none of the unprocessed points have a core distance anymore (i.e. aren't core points). These are

the outliers.