Clustering
DBSCAN

M. R. Hasan

CSCE 411/811

Data Modeling for Systems Development

Readings

- "*A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*" by Martin Ester, Hans-Peter Kriegel, Jiirg Sander, Xiaowei Xu
  https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf
- https://en.wikipedia.org/wiki/DBSCAN

What We Will Cover

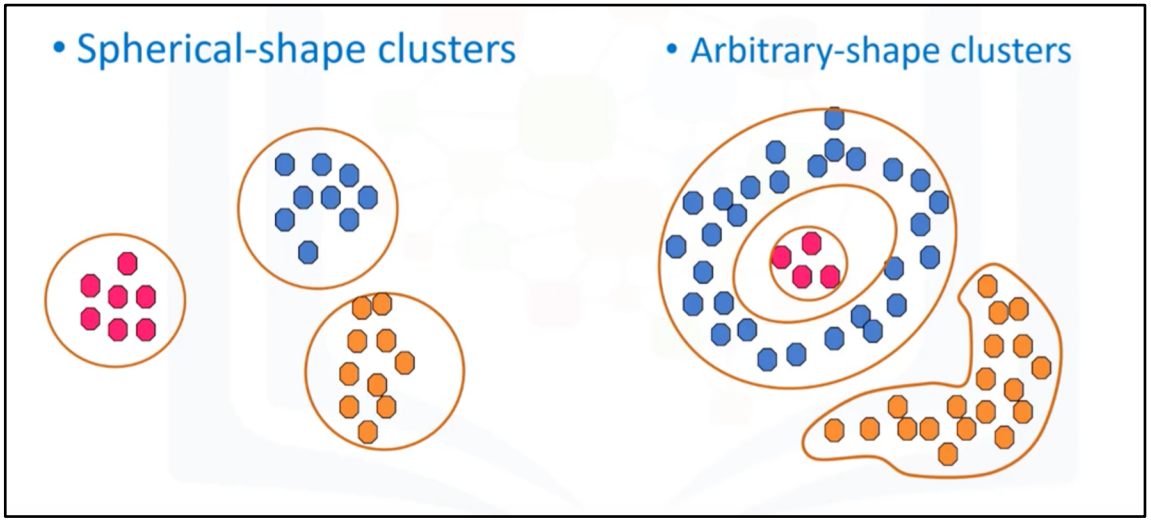
- Clustering
- DBSCAN

Effective Clustering on Large Dataset

- An effective clustering algorithm on a **large dataset** needs to fulfill **two criteria**.

  - Minimal requirements of **domain knowledge** to determine the input parameters (e.g., number of clusters).

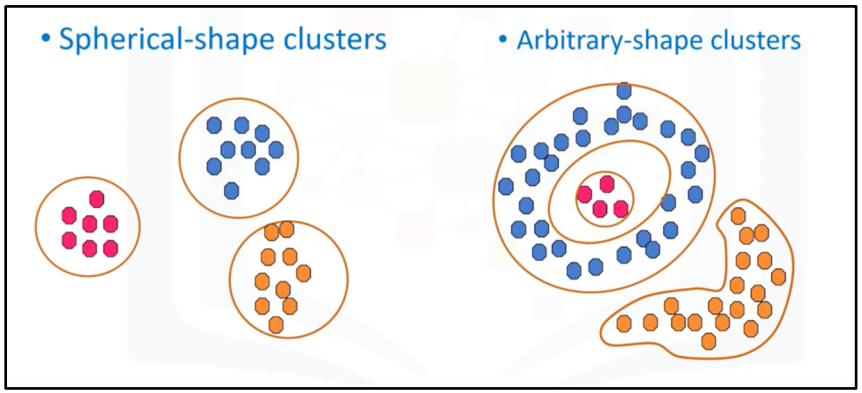
  - Discovery of clusters with **arbitrary shape** and good efficiency on large dataset.



Effective Clustering on Large Database

- The well-known clustering algorithms, such as the K-Means and GMM, offer **no solution** to the combination of these requirements.
- To fulfill these requirements, the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) clustering algorithm is designed.

- Minimal requirements of **domain knowledge** to determine the input parameters (e.g., number of clusters).
- Discovery of clusters with **arbitrary shape** and good efficiency on large databases.

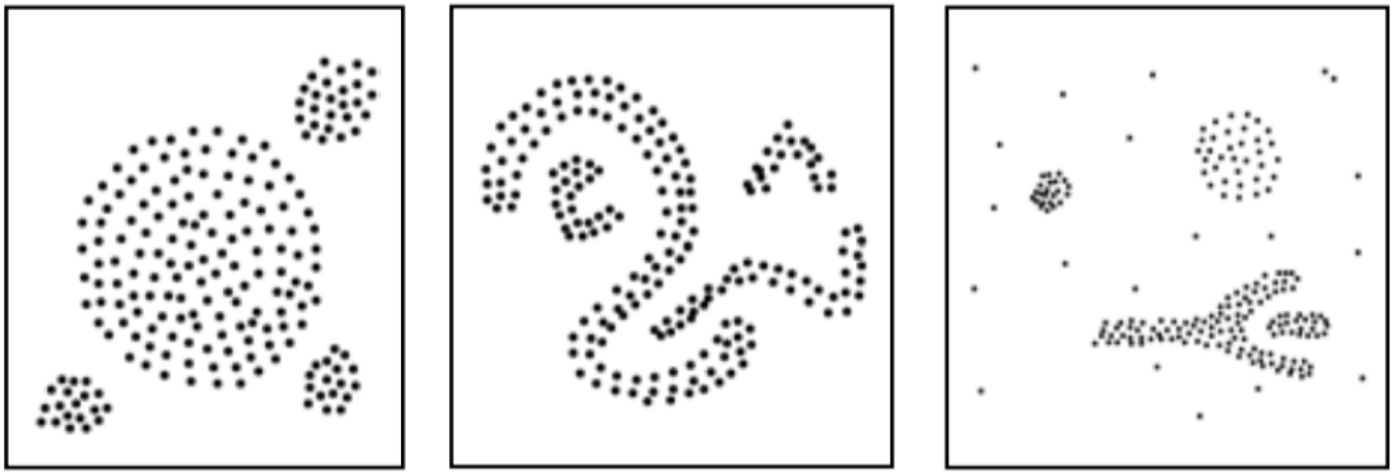


DBSCAN

- DBSCAN relies on a **density-based notion of clusters**.
- It is designed to **discover clusters of arbitrary shape**.

DBSCAN

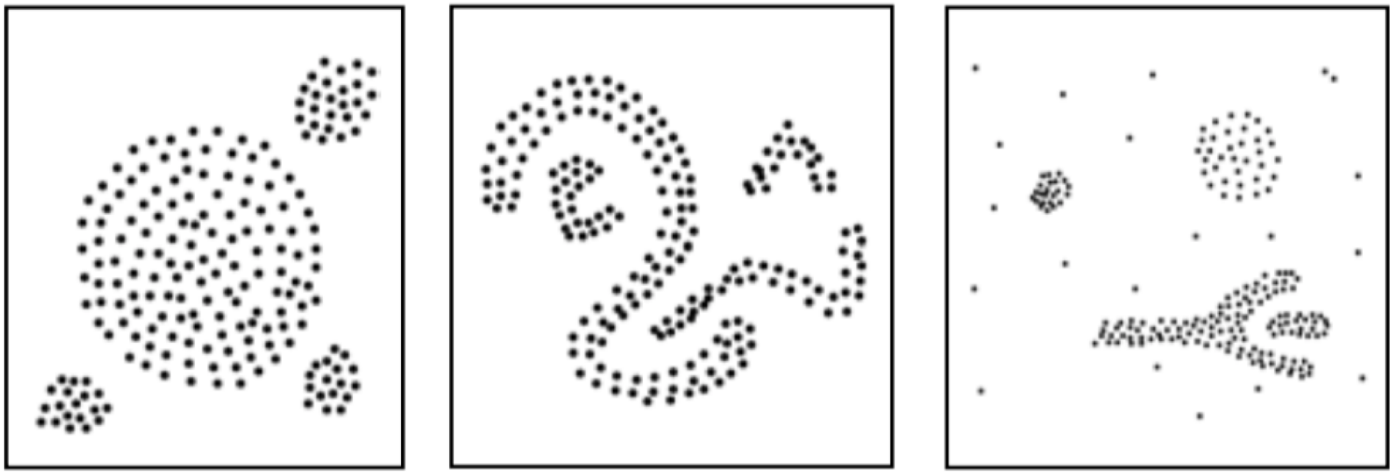
- Let's explain the **Density Based Notion of Clusters**.
- Consider the sample sets of points.
- We can **easily and unambiguously detect clusters** of points and **noise points** not belonging to any of those clusters.



database 1 **database 2** **database 3**

DBSCAN

- The **main reason** why we recognize the clusters is:
- Within each cluster we have a **typical density of points** which is **considerably higher** than outside of the cluster.
- Furthermore, the density within the areas of ***noise is lower*** than the density in any of the clusters.



database 1 **database 2** **database 3**

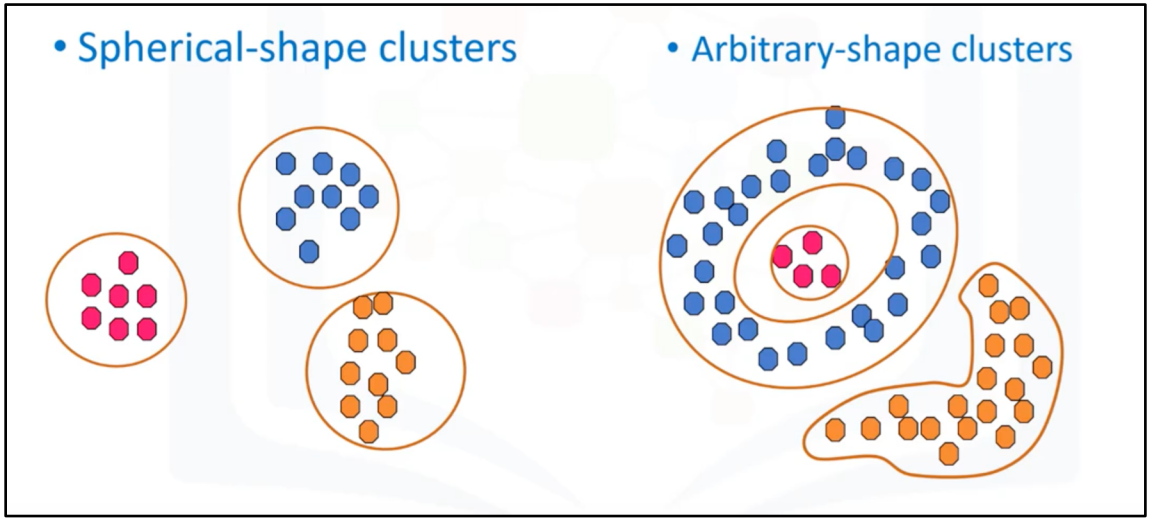
DBSCAN

- DBSCAN is the most well-known **density-based** clustering algorithm.
- It was first introduced in ***1996 by Ester et. al***.
- https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf
- Due to its importance in both theory and applications, this algorithm is one of three algorithms awarded the **Test of Time Award at SIGKDD 2014**.
- https://www.kdd.org/News/view/2014-sigkdd-test-of-time-award

DBSCAN

- Unlike K-Means & GMM, DBSCAN *does not require* the number of clusters as a parameter.
- Rather it **infers the number of clusters** based on the data.

DBSCAN can discover clusters of **arbitrary shape**.



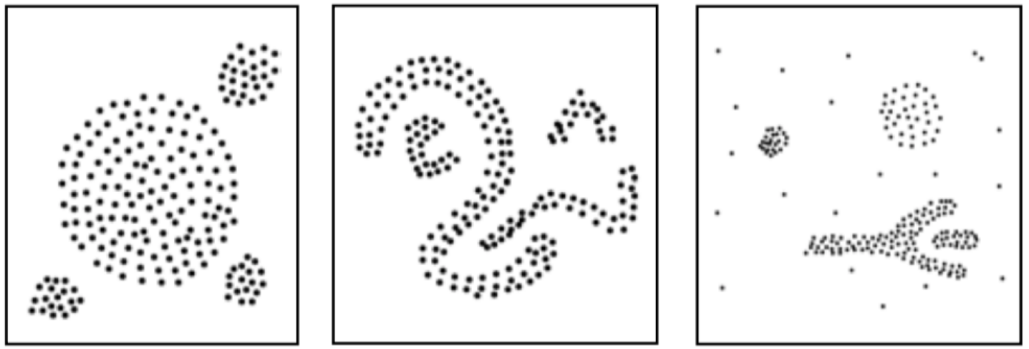
DBSACN: Preliminary

DBSCAN

- The DBSCAN algorithm defines clusters as continuous regions of high density.
- The DBSCAN algorithm has **two parameters**:

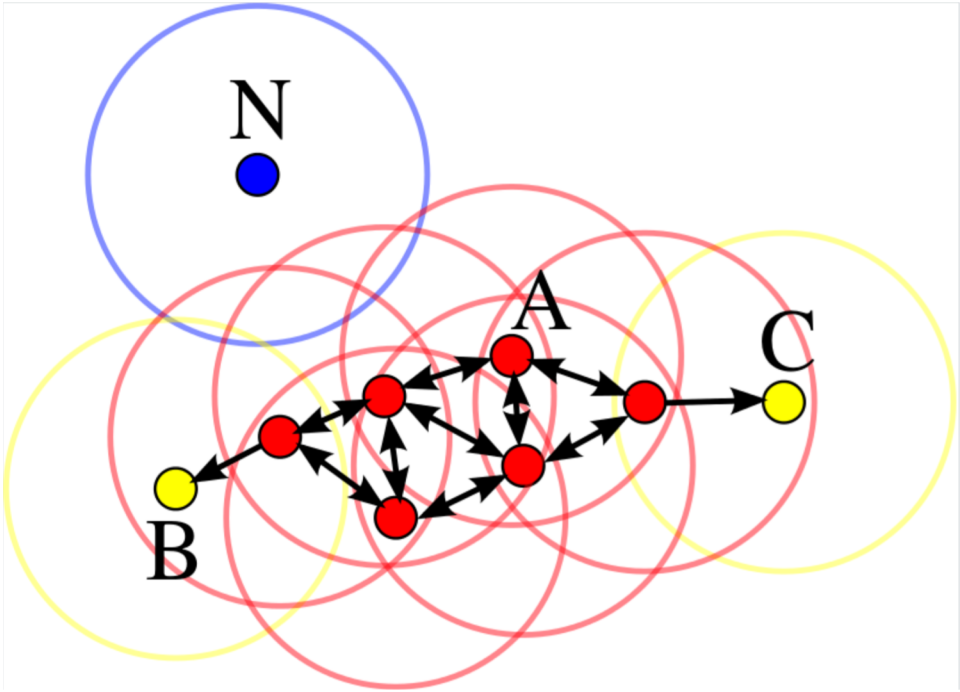
  - ε: The radius of the neighborhoods around a data point.

  - minPts: The minimum number of data points we want in a neighborhood to define a cluster.



DBSCAN

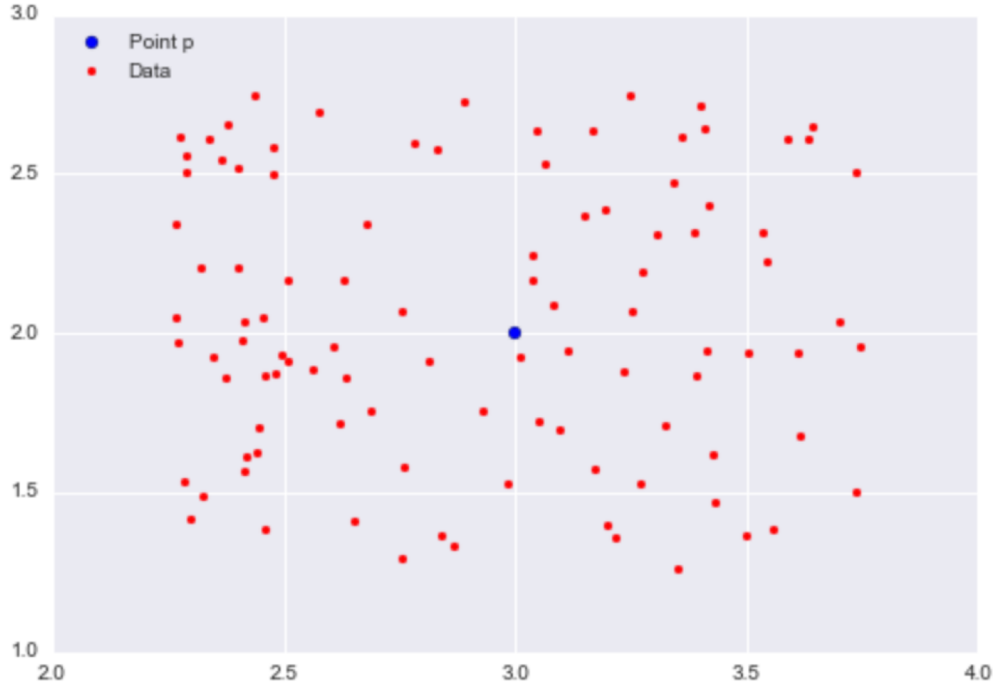
- For each instance, the algorithm counts **how many instances are located** within a small distance ε (epsilon) from it.
- This region is called the instance's ε-neighborhood.
- Let's explain this.



DBSCAN

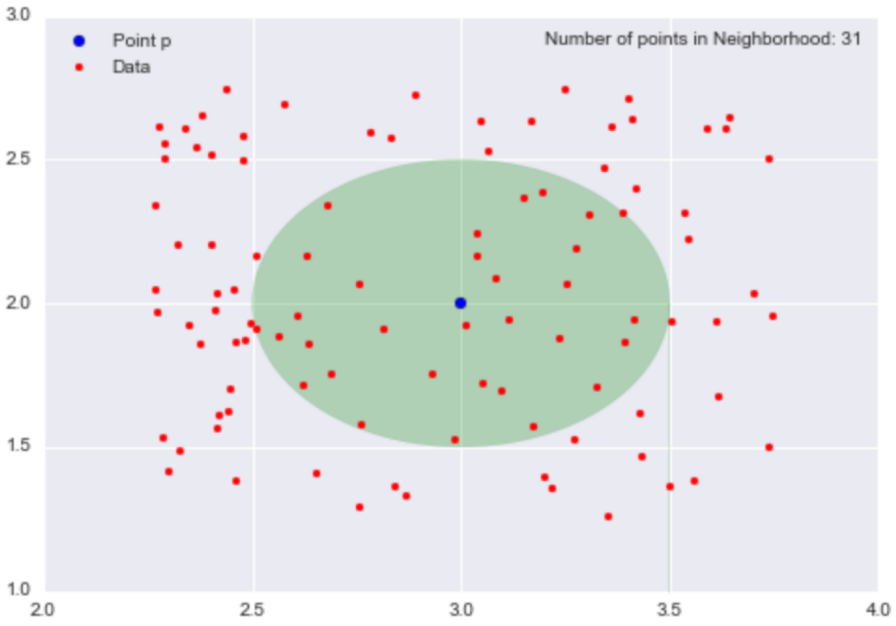
- Consider a set of points in some space to be clustered.
- Let ***epsilon ε*** be a parameter **specifying the radius** of a neighborhood with respect to some point (e.g., blue point).

Two points are considered **neighbors** if the distance between the two points is *below the threshold epsilon*.

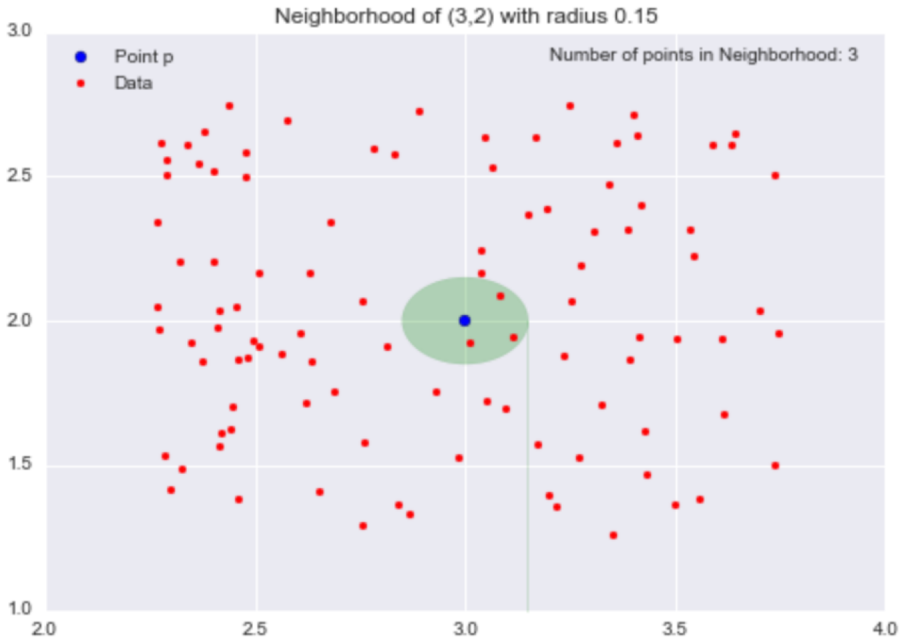


DBSCAN

- Two points are considered neighbors if the distance between the two points is *below the threshold epsilon*.



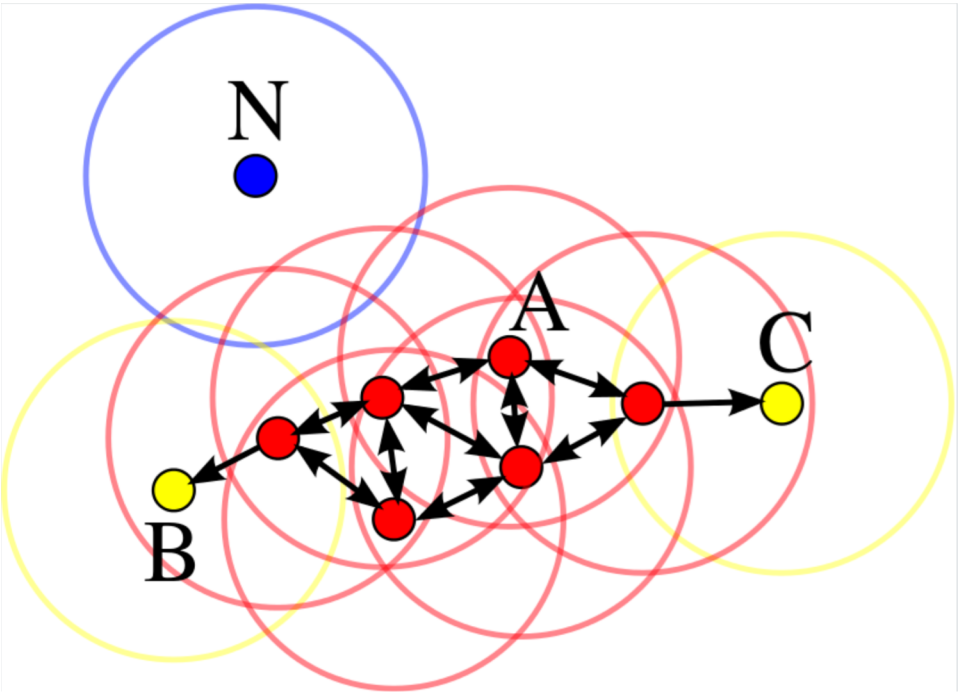
Epsilon=0.5



Epsilon=0.15

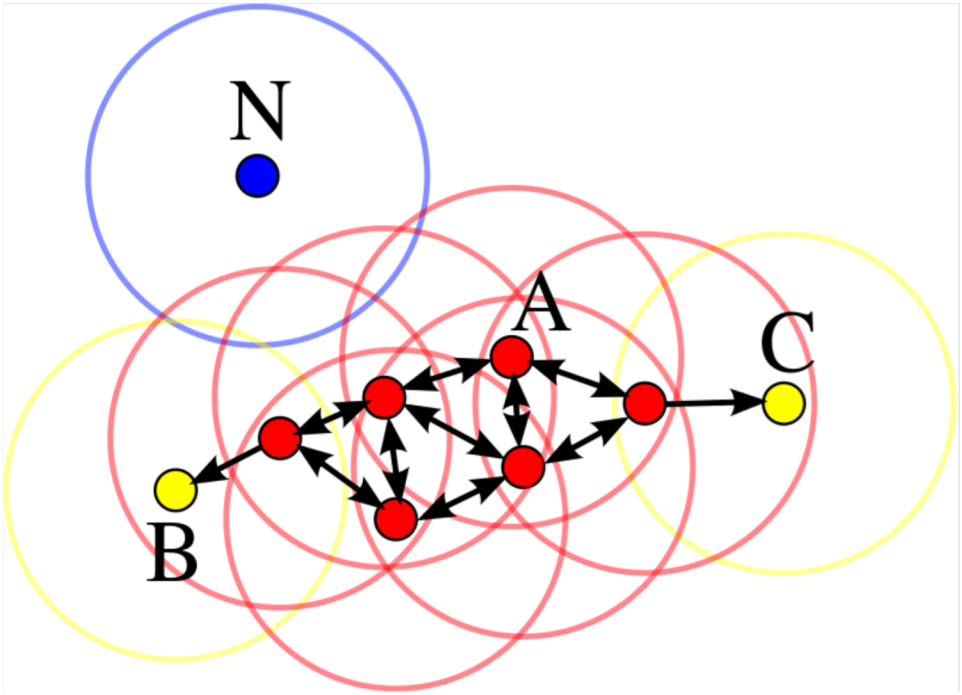
DBSCAN

- DBSCAN algorithm is based on **three types of points**.
- Core points
- Density-reachable points
- Outliers (noise points)



DBSCAN

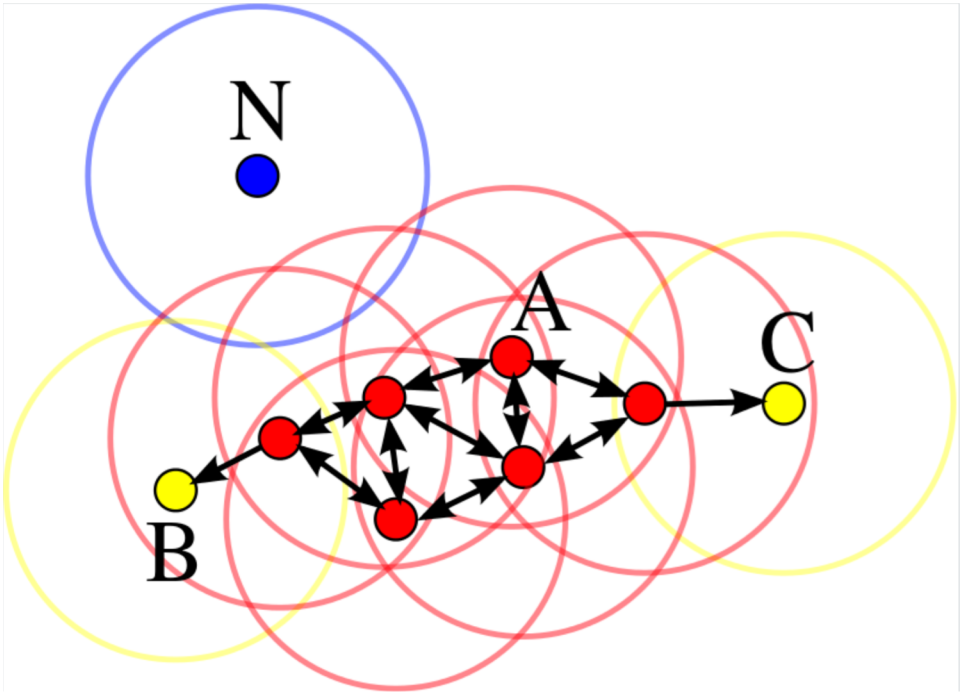
- A point is a **core point** if at least *minPts* number of points are within distance ε of it (including that point).
- The minPts points are the **minimum number of neighbors** a given point should have in order to be classified as a core point.



DBSCAN

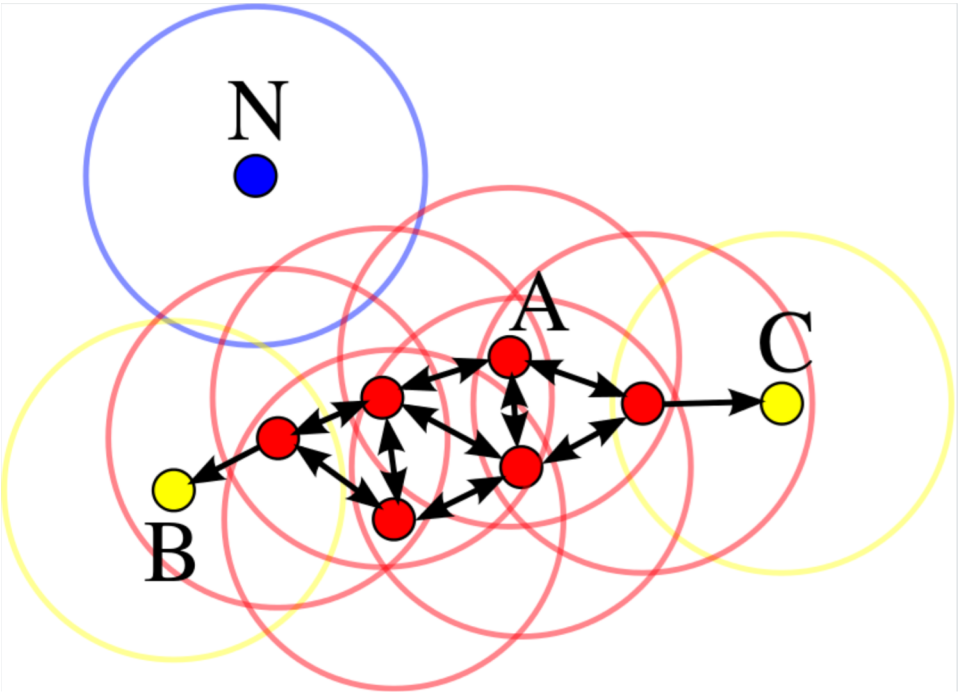
- Core Point (minPts = 4)

- Point *A* and the other **red points** are core points.
- Because the area surrounding these points in an ε radius contain **at least 4 points** (including the point itself).



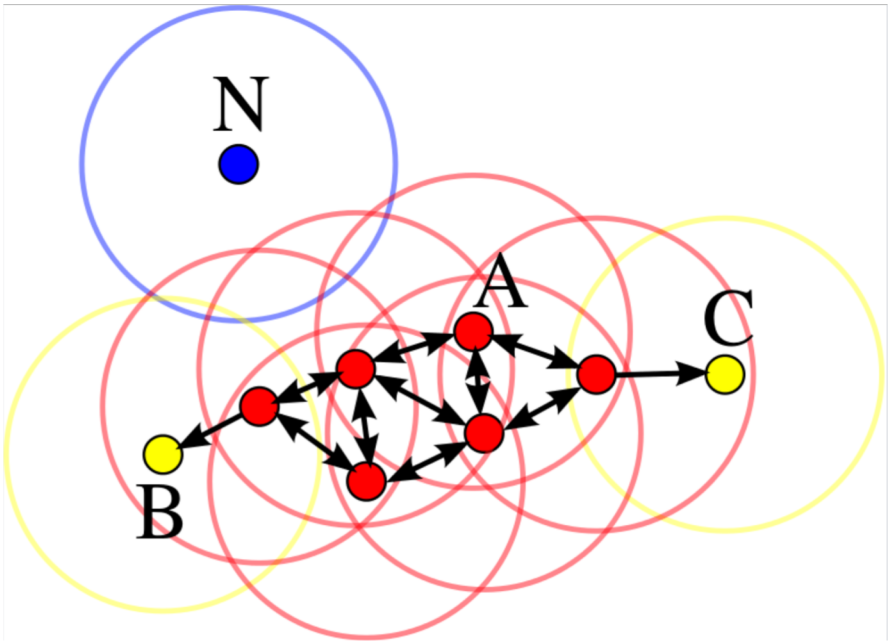
DBSCAN

- In other words, core points are those that are **located in dense regions**.
- The **point itself is included** in the minimum number of samples.



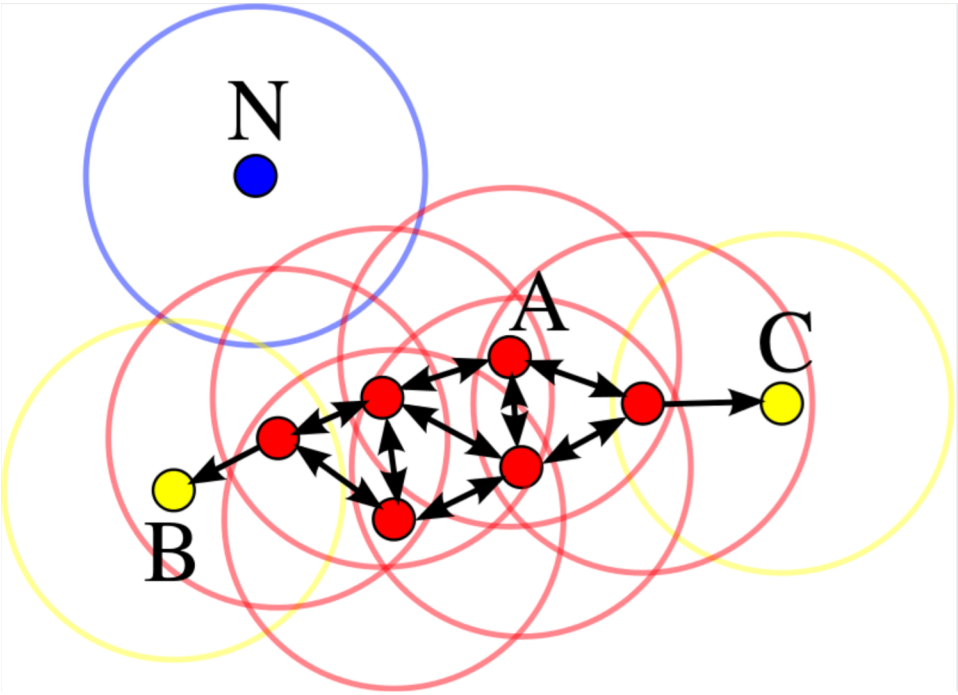
DBSCAN

- The **clusters are built** around the core points.
- So by adjusting the minPts parameter, we can **fine-tune how dense our clusters** must be.



DBSCAN

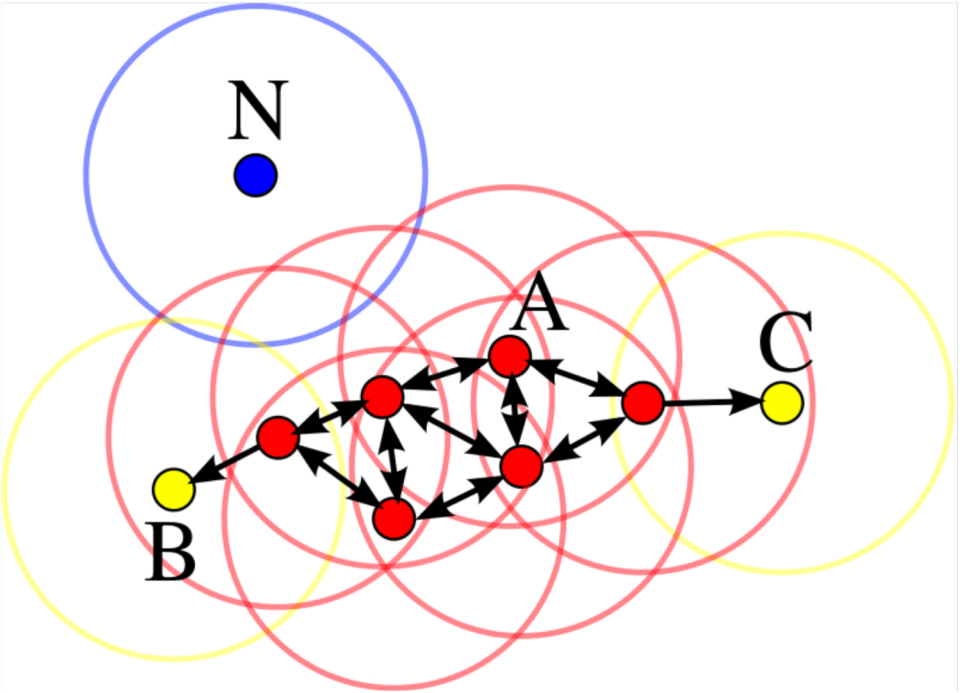
- All instances in the neighborhood of a core point belong to the **same cluster**.
- This neighborhood may include **other core points**.
- Therefore, **a long sequence** of neighboring core points forms a single cluster (red points).



DBSCAN

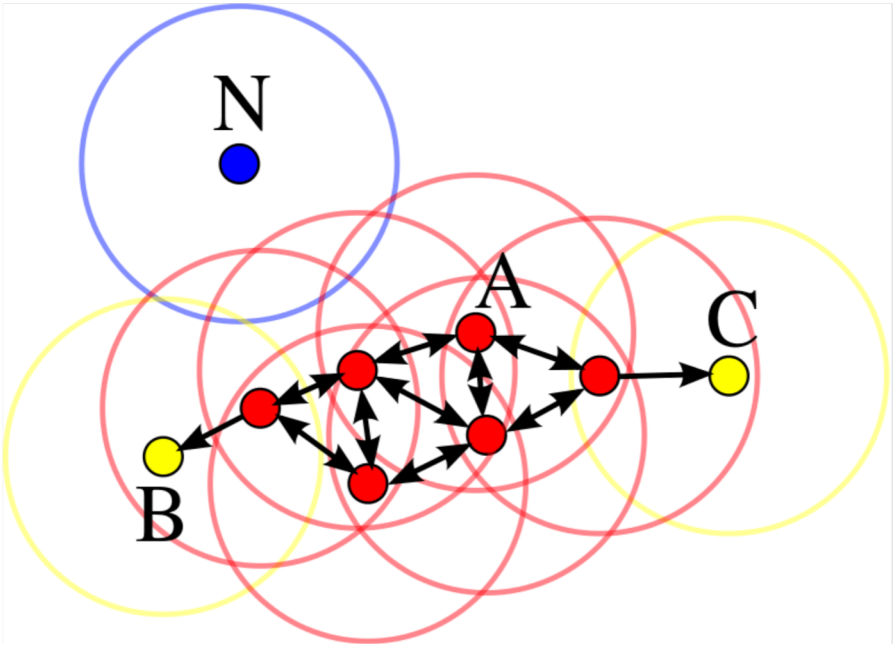
- If A is a core point, then it forms a cluster together with all points (core or non-core) that are **reachable from it**.
- Each cluster contains at least one core point.

Non-core points can be part of a cluster, but they form its "**border**", since they cannot be used to reach more points.



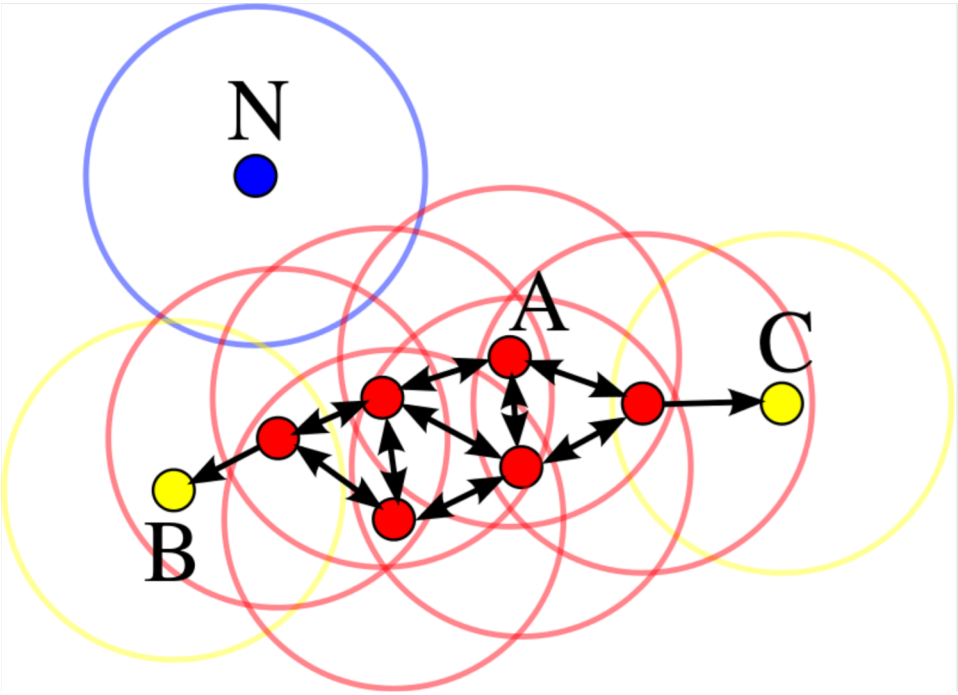
DBSCAN

- The yellow points (B & C) are **non-core** or **border points**.
- These points are within epsilon radius of core points but **do not contain minimum number of neighbors**.



DBSCAN

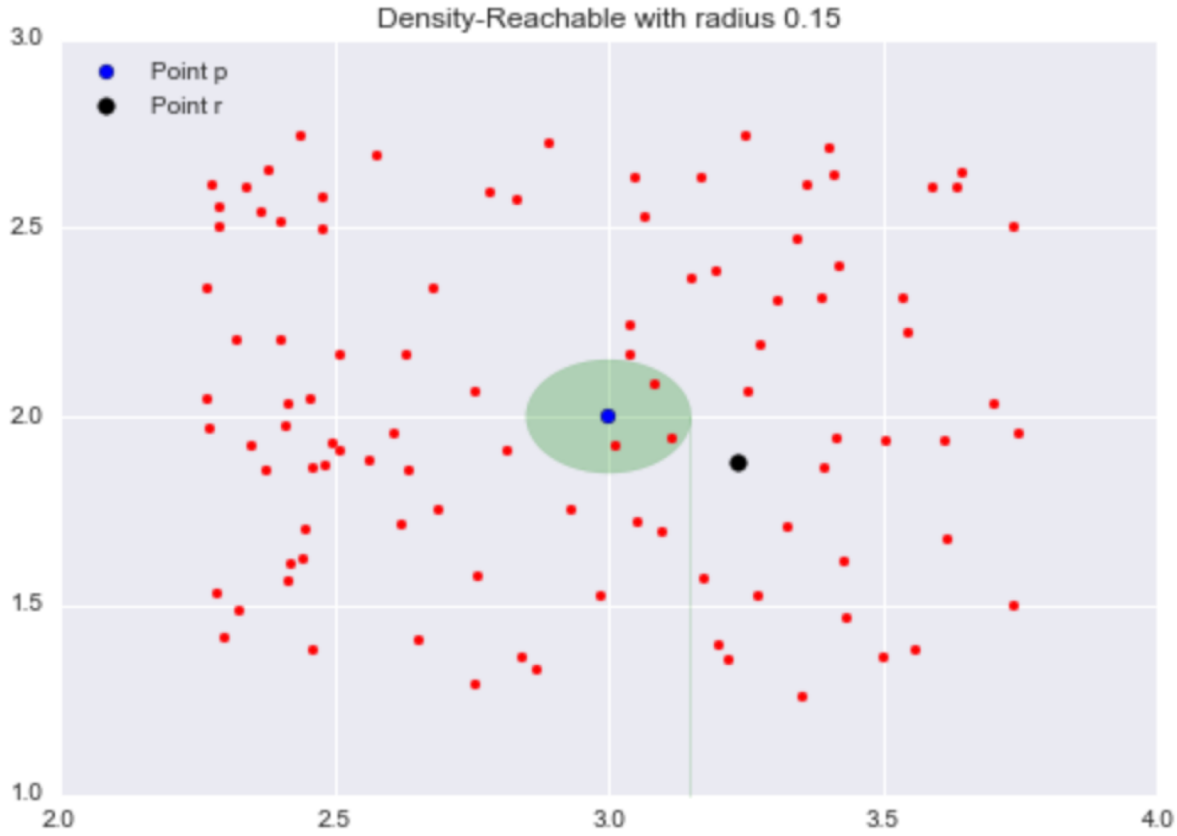
- To better comprehend the border points we need to understand the concept of **density-reachable**.



DBSCAN

- Consider the neighborhood of the point ***p*** (blue point) with epsilon = 0.15.

Consider the **point r (black dot)** that is outside of the point ***p***'s neighborhood.

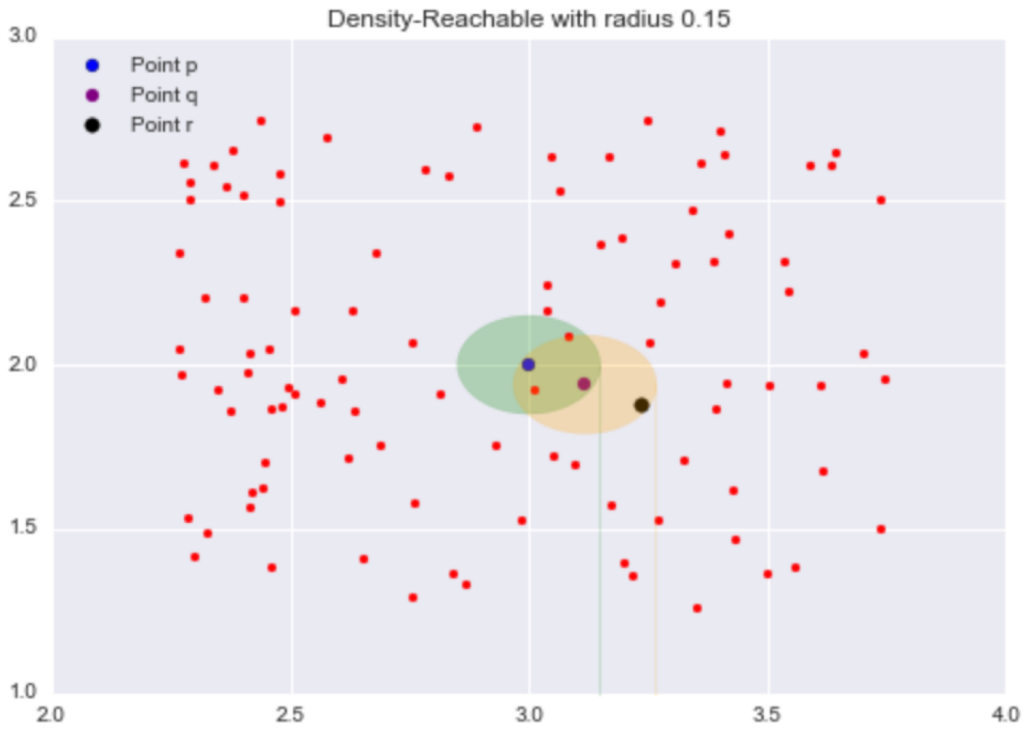


DBSCAN

- All the points inside the point p's neighborhood are said to be **directly reachable from p**.

Let's explore the **neighborhood of point q**, a point directly reachable from p.

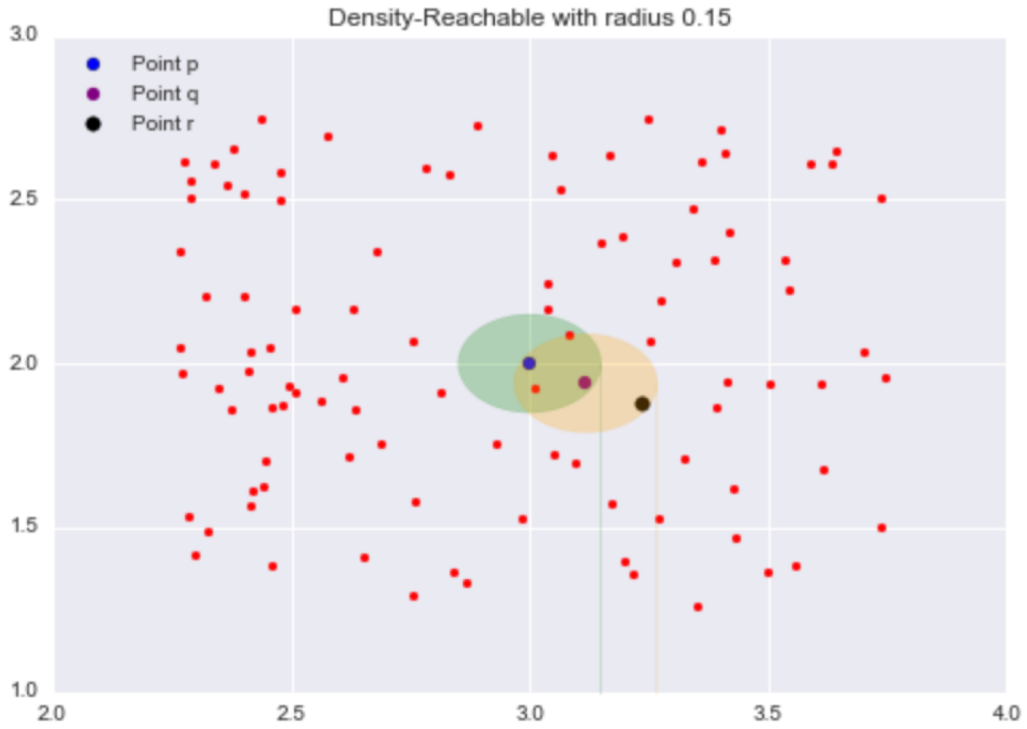
The **yellow circle** represents q's neighborhood.



DBSCAN

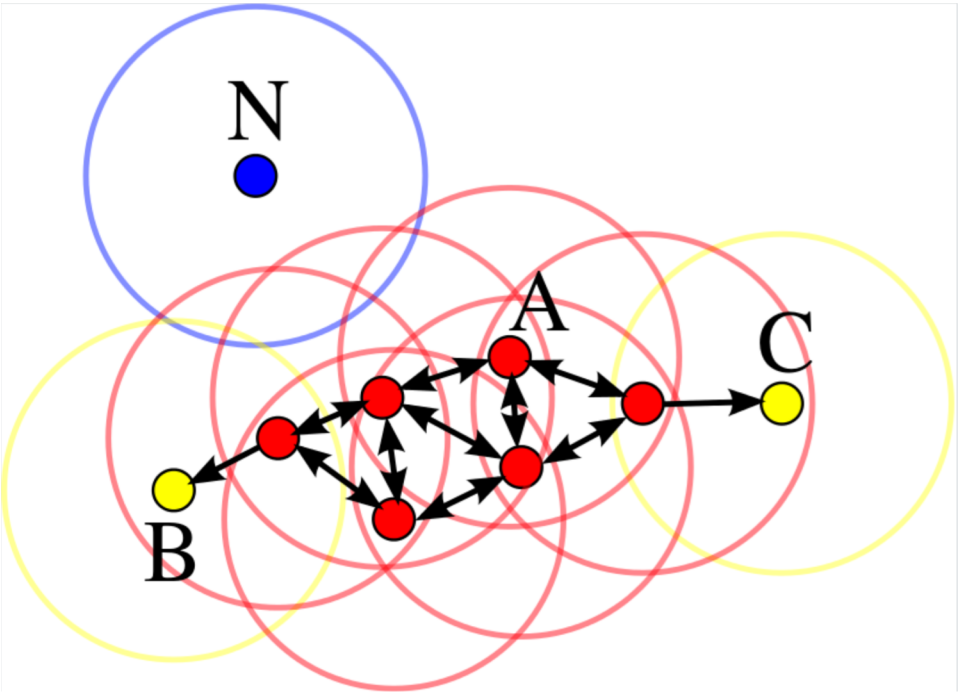
- The **target point r** is not in our starting point p's neighborhood.
- It is contained in the point q's neighborhood.

If we can get to the point r by *jumping from neighborhood to neighborhood*, starting at a point p, then the point r is **density-reachable** from the point p.



DBSCAN

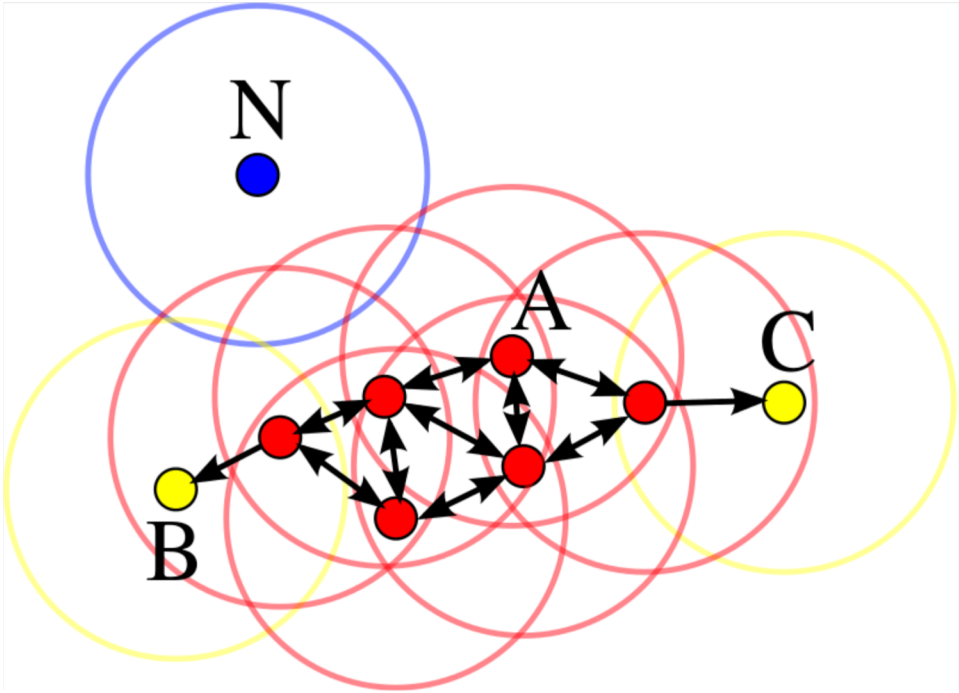
- We can think of density-reachable points as being the "**friends of a friend**".



DBSCAN

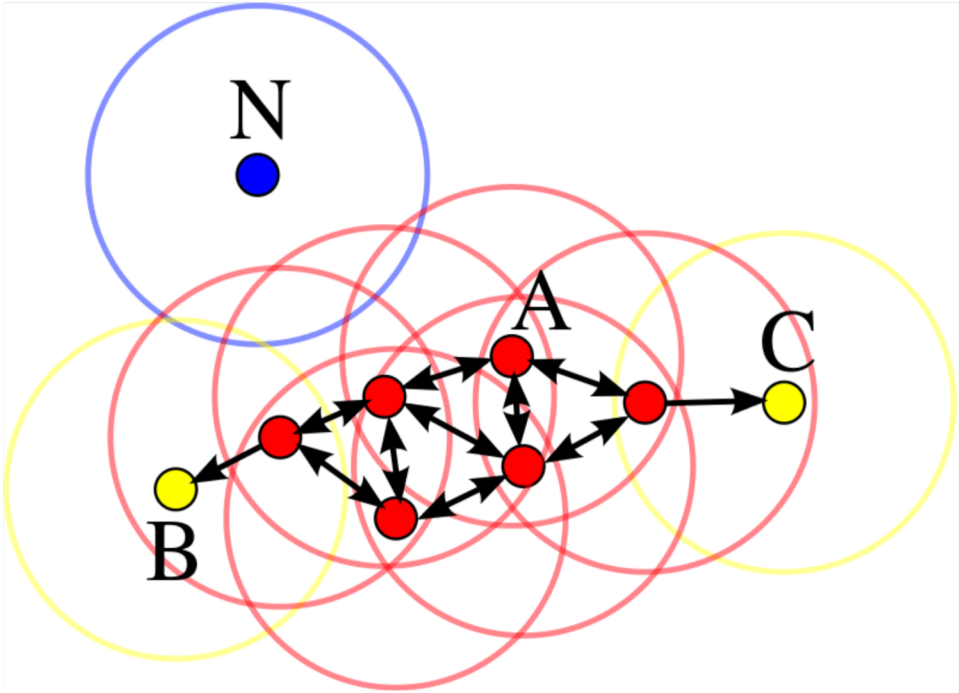
- Density-reachable points is ***not limited*** to just two adjacent neighborhood jumps.
- As long as we can reach the point doing "**neighborhood jumps**", starting at a core point p, that point is density-reachable from p.

Thus, "*friends of a friend of a friend … of a friend*" are included as well.



DBSCAN

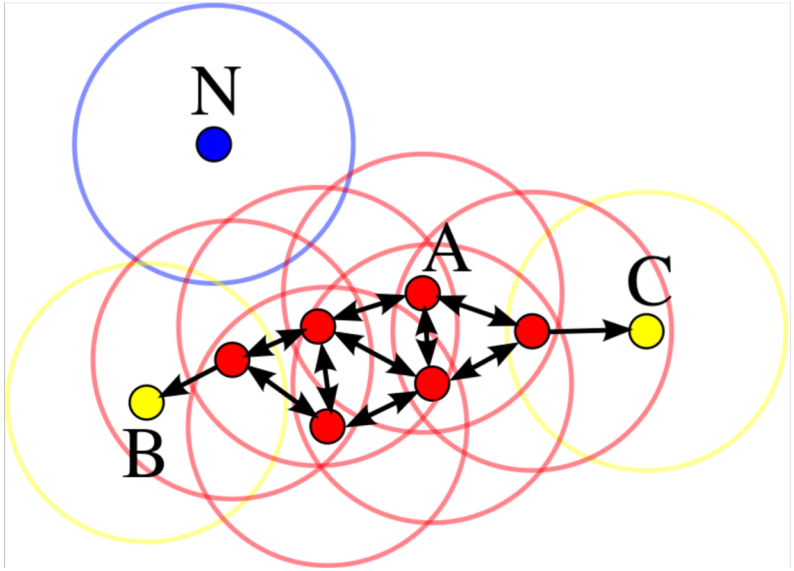
- The idea of density-reachable is **dependent** on the value of ε.
- By picking larger values of ε, more points become density-reachable.
- By choosing smaller values of ε, less points become density-reachable.



DBSCAN

- So far we introduced Core points (A) & Density-Reachable points (B & C)
- Any instance that is **not a core point** and nor are they close enough to a cluster to be density-reachable from a core point is considered a **noise/anomaly/outlier** (blue point, N).

- DBSCAN algorithm is based on **three types of points**.
- Core points
- Density-reachable points
- Outliers (noise points)



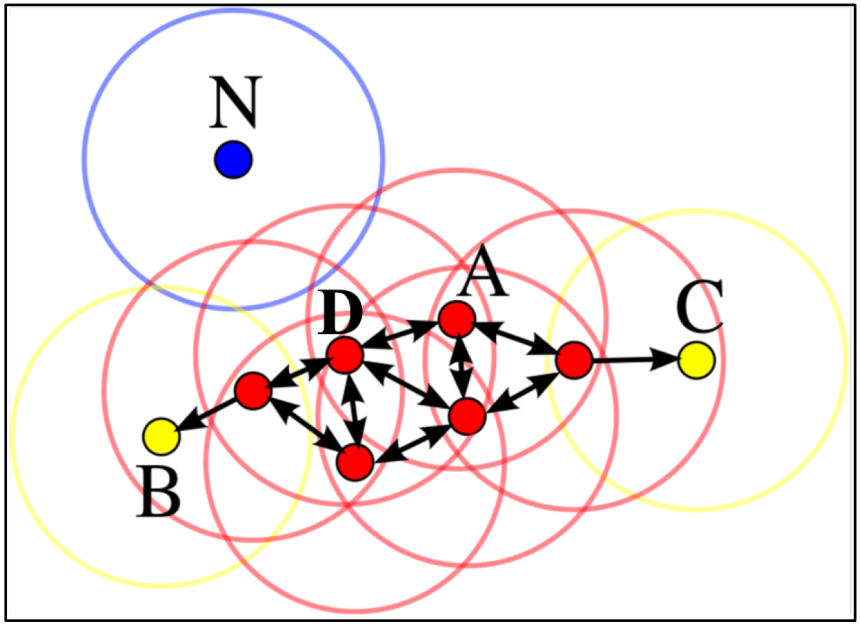
DBSACN: Algorithm

DBSCAN

- DBSCAN requires **two parameters**:

  - ε (epsilon): The radius of the neighborhoods around a data point.

  - minPts: the minimum number of points required to form a dense region.

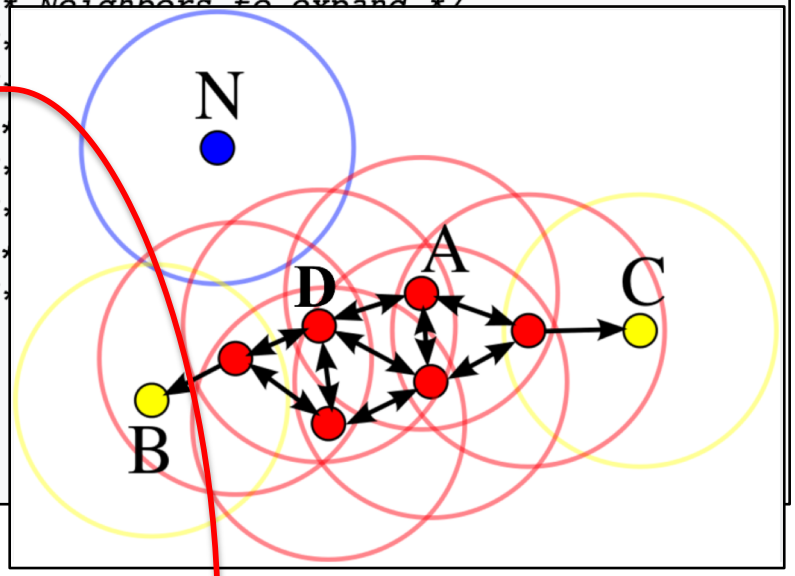


```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue              /* Previously processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                             /* Density check */
            label(P) = Noise                               /* Label as Noise */
            continue
        }
        C = C + 1                                          /* next cluster label */
        label(P) = C                                       /* Label initial point */
        Seed set S = N \ {P}                               /* Neighbors to expand */
        for each point Q in S {                            /* Process every seed point */
            if label(Q) = Noise then label(Q) = C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue          /* Previously processed */
            label(Q) = C                                   /* Label neighbor */
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                         /* Density check */
                S = S ∪ N                                  /* Add new neighbors to seed set */
            }
```

The algorithm begins with an **arbitrary starting point** that has ***not been visited***.

This point's ε-neighborhood is retrieved.

```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue              /* Previously processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                             /* Density check */
            label(P) = Noise                               /* Label as Noise */
            continue
        }
        C = C + 1                                          /* next cluster label */
        label(P) = C                                       /* Label initial point */
        Seed set S = N \ {P}                               /* Neighbors to expand */
        for each point Q in S {                            /*
            if label(Q) = Noise then label(Q) = C          /*
            if label(Q) ≠ undefined then continue          /*
            label(Q) = C                                   /*
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /*
            if |N| ≥ minPts then {                         /*
                S = S ∪ N                                  /*
            }
```



The algorithm begins with an **arbitrary starting point** that has ***not been visited***.

This point's ε-neighborhood is retrieved.

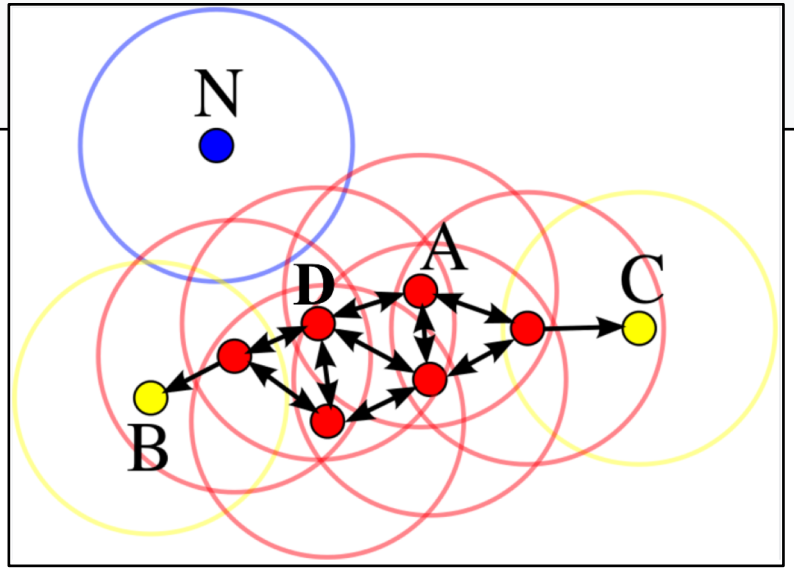
If it ***does not*** contain sufficiently many points (e.g., C), it's labeled as **noise**.

Note that a noise point might **later be found** in a sufficiently sized ε-environment of a different point and hence **be made part of a cluster**.

```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue              /* Previously processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                             /* Density check */
            label(P) = Noise                               /* Label as Noise */
            continue
        }
        C = C + 1                                          /* next cluster label */
        label(P) = C                                       /* Label initial point */
        Seed set S = N \ {P}                               /* Neighbors to expand */
        for each point Q in S {                            /* Process every seed point */
            if label(Q) = Noise then label(Q) = C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue          /* Previously processed */
            label(Q) = C                                   /* Label neighbor */
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                         /* Density check */
                S = S ∪ N                                  /* Add new neighbors to seed set */
            }
        }
    }
}
```

If it contains sufficiently many points (e.g. A), **a cluster is started**.

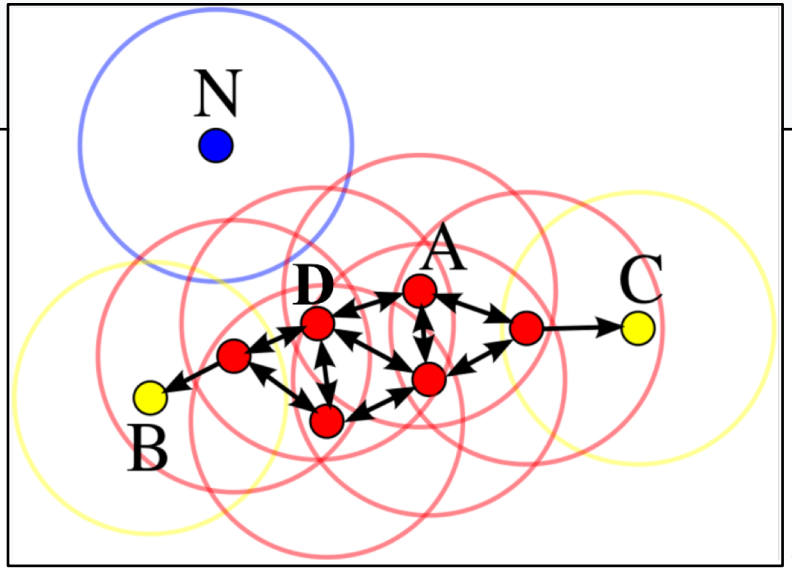
Expand the cluster by adding all directly-reachable points to the cluster.



```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue              /* Previously processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                             /* Density check */
            label(P) = Noise                               /* Label as Noise */
            continue
        }
        C = C + 1                                          /* next cluster label */
        label(P) = C                                       /* Label initial point */
        Seed set S = N \ {P}                               /* Neighbors to expand */
        for each point Q in S {                            /* Process every seed point */
            if label(Q) = Noise then label(Q) = C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue          /* Previously processed */
            label(Q) = C                                   /* Label neighbor */
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                         /* Density check */
                S = S ∪ N                                  /* Add new neighbors to seed set */
            }
        }
    }
}
```

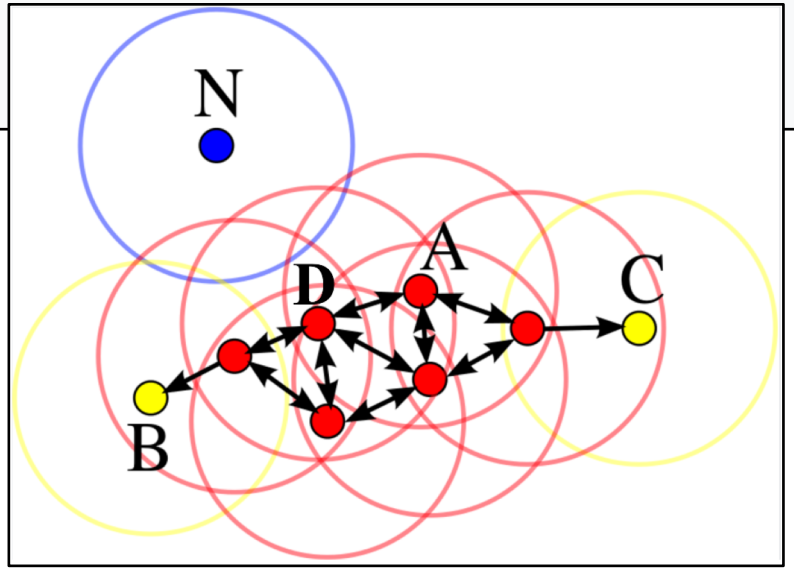
If a point is found to be in a dense part of a cluster (e.g., D is found in the **dense** ε-neighborhood of A), it becomes a part of that **cluster**.

Also the ε-neighborhood of that point (e.g., D) is added to the cluster, ***when they are also dense*** (i.e., has enough neighbors)



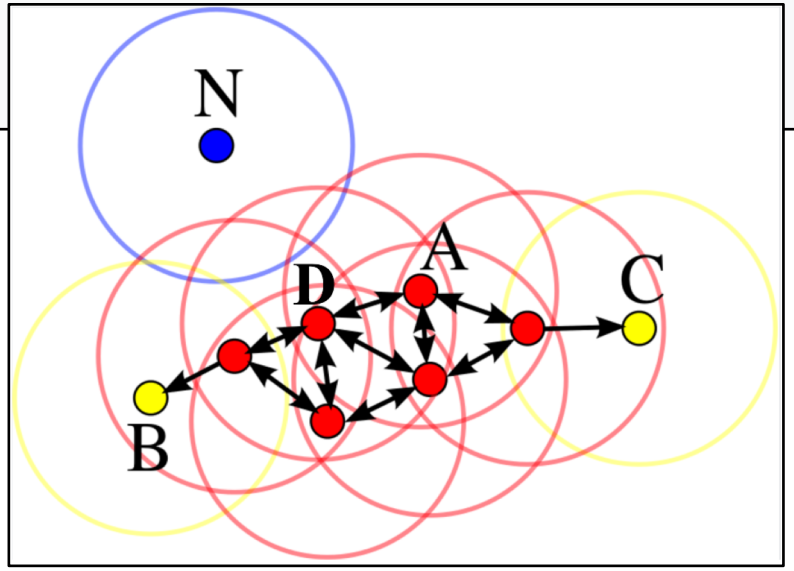
```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                   /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)      /* Find neighbors */
        if |N| < minPts then {                              /* Density check */
            label(P) = Noise                                /* Label as Noise */
            continue
        }
        C = C + 1                                           /* next cluster label */
        label(P) = C                                        /* Label initial point */
        Seed set S = N \ {P}                                /* Neighbors to expand */
        for each point Q in S {                             /* Process every seed point */
            if label(Q) = Noise then label(Q) = C           /* Change Noise to border point */
            if label(Q) ≠ undefined then continue           /* Previously processed */
            label(Q) = C                                    /* Label neighbor */
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                          /* Density check */
                S = S ∪ N                                   /* Add new neighbors to seed set */
            }
        }
    }
}
```

This **process continues** until the density-connected cluster is completely found.



```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                   /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)      /* Find neighbors */
        if |N| < minPts then {                              /* Density check */
            label(P) = Noise                                /* Label as Noise */
            continue
        }
        C = C + 1                                           /* next cluster label */
        label(P) = C                                        /* Label initial point */
        Seed set S = N \ {P}                                /* Neighbors to expand */
        for each point Q in S {                             /* Process every seed point */
            if label(Q) = Noise then label(Q) = C           /* Change Noise to border point */
            if label(Q) ≠ undefined then continue           /* Previously processed */
            label(Q) = C                                    /* Label neighbor */
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                          /* Density check */
                S = S ∪ N                                   /* Add new neighbors to seed set */
            }
        }
    }
}
```

Then, a ***new unvisited point*** is retrieved and processed, leading to the discovery of a further cluster or noise.



```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue              /* Previously processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                             /* Density check */
            label(P) = Noise                               /* Label as Noise */
            continue
        }
        C = C + 1                                          /* next cluster label */
        label(P) = C                                       /* Label initial point */
        Seed set S = N \ {P}                               /* Neighbors to expand */
        for each point Q in S {                            /* Process every seed point */
            if label(Q) = Noise then label(Q) = C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue          /* Previously processed */
            label(Q) = C                                   /* Label neighbor */
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                         /* Density check */
                S = S ∪ N                                  /* Add new neighbors to seed set */
            }
        }
    }
}
```

```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors = 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 */
            Neighbors = Neighbors ∪ {P}                /* Add to result */
        }
    }
    return Neighbors
}
```

DBSACN: Setting Two Key Parameters

DBSCAN

- The DBSCAN algorithm has **two parameters**:

  - ε: The radius of the neighborhoods around a data point.

  - minPts: The minimum number of data points we want in a neighborhood to define a cluster.

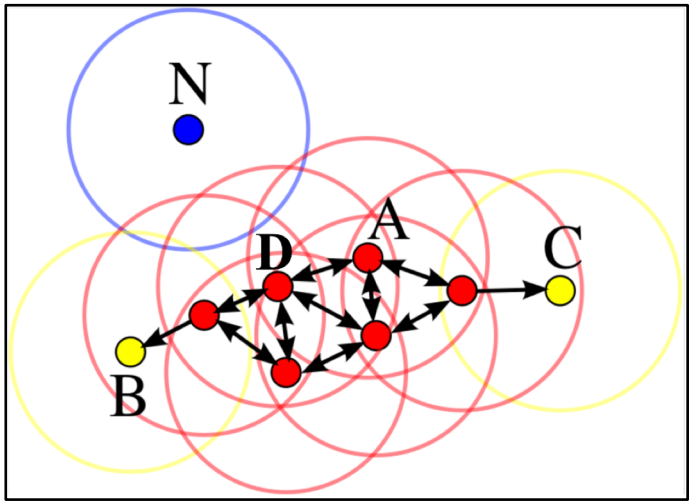
Setting the values of these two parameters is ***very tricky***.

DBSCAN

- There is **no automated way** to set these values.
- We need to decide **based on the kind of data** that we are using.

- We provide a ***guideline*** for setting the two parameters.

DBSCAN

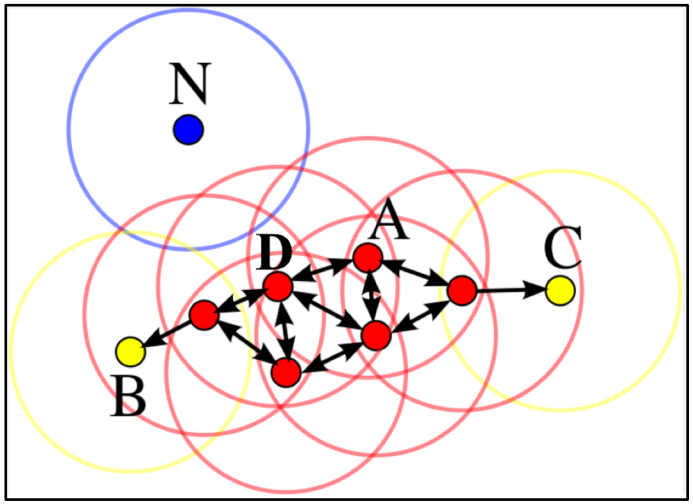


- epsilon:
- ***Very small value***: There wouldn't be enough points in a region to form a cluster.

  - It would make **most of the points outliers**.
- ***Very large value***: The majority of the points would fall into one cluster.

  - There would be almost **no outliers**.
- Thus, we need to select the value **wisely**.
- More often or not, a smaller value is preferred for epsilon.

DBSCAN



- minPts:
- ***Very large value***:

  - It would need more points to form a cluster.

  - Thus, it would leave a ***major chunk of points as outliers***.

  - Example: if minPts = 6, all points are outliers.

- ***Very small value***:

  - It would make clusters form even for what could have been an outlier.

  - If minPts = 1, there is no outlier.

DBSACN: Complexity

DBSCAN

- DBSCAN **visits each point of the dataset**, possibly multiple times (e.g., as candidates to different clusters).
- Thus, the ***time complexity is mostly governed*** by the number of RangeQuery invocations.

```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors = 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 */
            Neighbors = Neighbors ∪ {P}                  /* Add to result */
        }
    }
    return Neighbors
}
```

DBSCAN

- Time-Complexity
- DBSCAN executes exactly one such query for each point.
- The **worst-case** runtime complexity is $O(n^2)$.
- It can be **improved** by avoiding distance calculations via indexing structure.

```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors = 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 */
            Neighbors = Neighbors U {P}                  /* Add to result */
        }
    }
    return Neighbors
}
```

DBSCAN

- Time-Complexity
- DBSCAN executes exactly one such query for each point.
- If an indexing structure is used, then complexity of executing **a single neighborhood query in O(logn)**.
- Thus, an overall average runtime complexity:

O(nlogn)

But this will require **more memory**.

```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors = 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 */
            Neighbors = Neighbors ∪ {P}                  /* Add to result */
        }
    }
    return Neighbors
}
```

DBSCAN

- Space-Complexity
- The distance matrix of size $(n^2 - n)/2$ can be materialized to avoid distance recomputations.
- But this needs $O(n^2)$ memory.
- Whereas a non-matrix based implementation of DBSCAN only needs $O(n)$ memory.

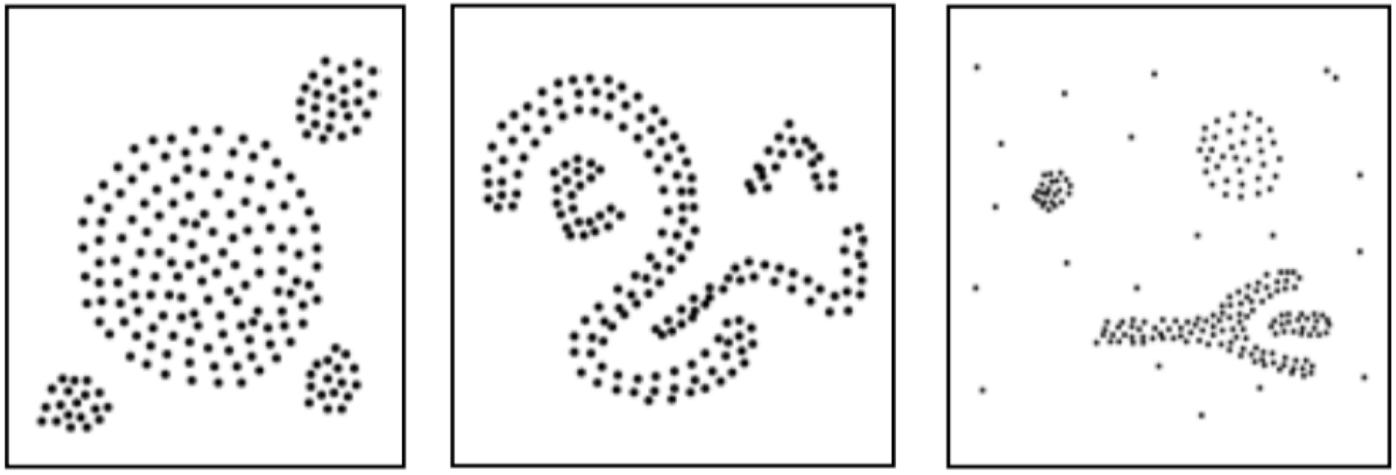
DBSACN: Advantages

DBSCAN

- DBSCAN being a density-based clustering algorithm is great at **separating clusters** of high density versus clusters of low density within a given dataset.
- DBSCAN can **sort data into clusters** of *varying shapes* as well, another strong advantage.

DBSCAN

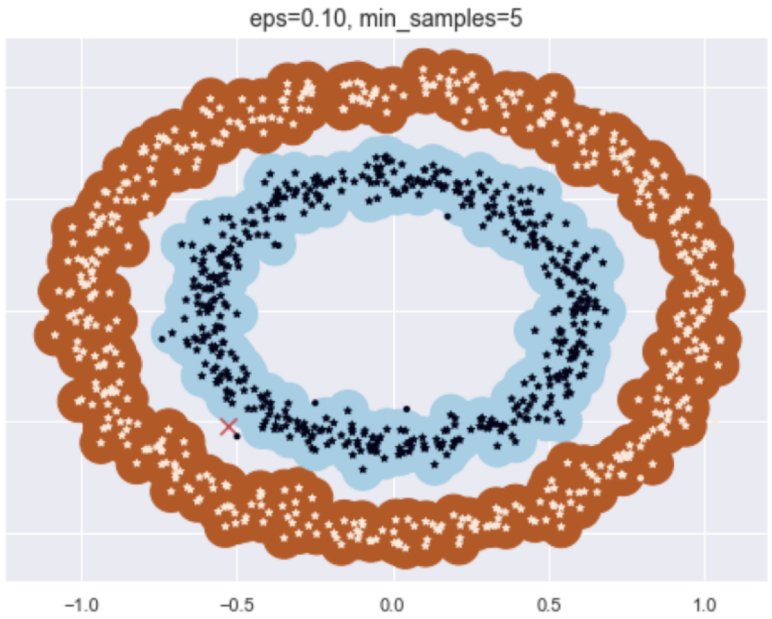
- It is great with **handling outliers** within the dataset.
- When forming a cluster, it is **less affected by the outliers**.
- Outliers are regions with low density as compared to the large density of the clusters.

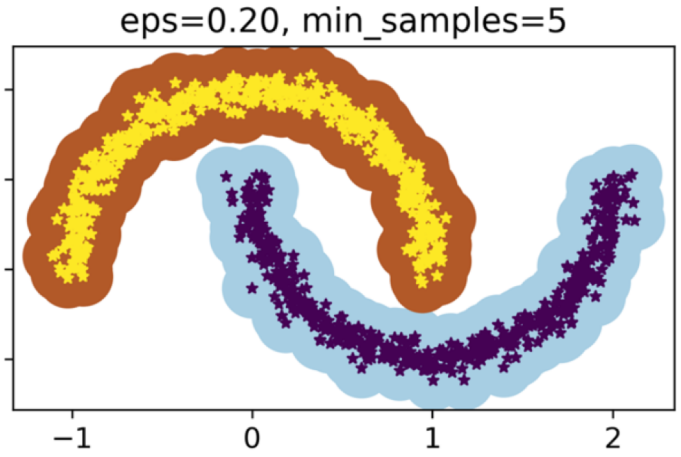


DBSCAN

- DBSCAN **does not require one to specify the number of clusters** in the data a priori, as opposed to K-Means.

DBSCAN can find **arbitrarily shaped** clusters.





It can even find a cluster **completely surrounded by** (but not connected to) a different cluster.

DBSCAN

- DBSCAN requires **just two parameters** and is mostly insensitive to the ordering of the points in the dataset.
- The parameters minPts and ε can be set by a **domain expert**, if the data is well understood.

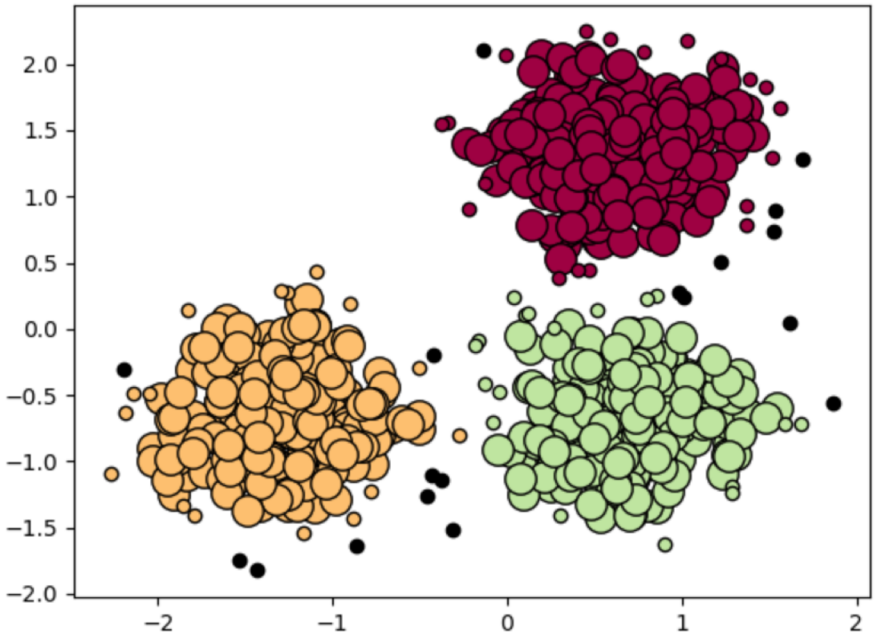
DBSACN: Disadvantages

DBSCAN

- The quality of DBSCAN **depends on the distance measure**.
- The most common distance metric used is **Euclidean distance**.
- Especially for high-dimensional data, this metric can be rendered almost useless due to the so-called "*Curse of dimensionality*", making it difficult to find an appropriate value for ε.

DBSCAN

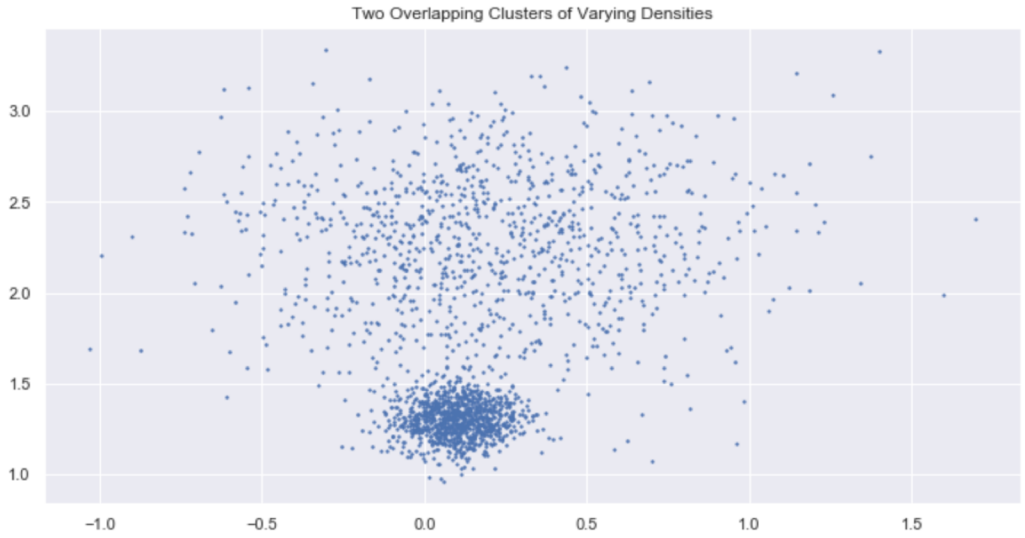
- The DBSCAN algorithm works well if:
  - All the clusters are **dense enough**.
  - Clusters are well **separated by low-density regions**.

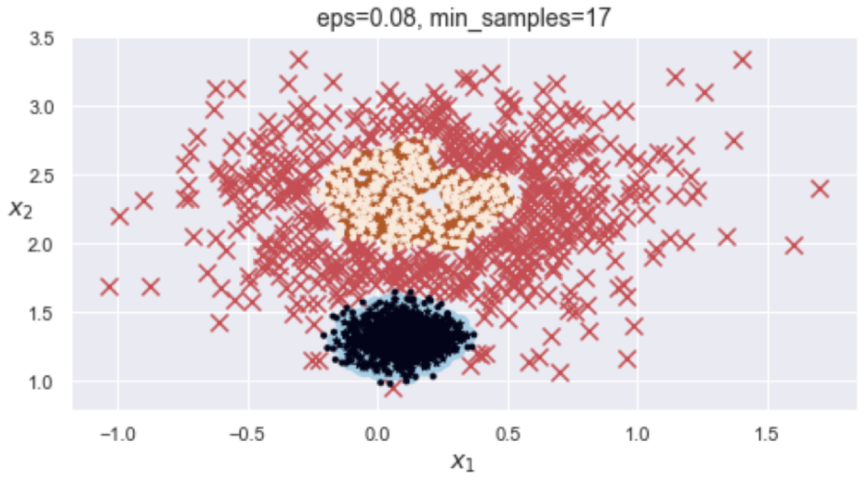


DBSCAN

- However, DBSCAN cannot cluster data sets well with **large differences in densities**.

  - This is because the minPts-ε combination cannot then be **chosen appropriately for all clusters**.





DBSCAN

- DBSCAN is **very sensitive to scale** since epsilon is a fixed value for the maximum distance between two points.
- If the **data and scale** are not well understood, choosing a meaningful distance threshold ε can be difficult.
- Thus, we must standardize the data before applying the DBSCAN algorithm.

DBSCAN

- There are situations where DBSCAN **performs well**, while sometimes its **performance is very bad**.

DBSCAN

- In general, when we **don't know the number of clusters hidden** in the dataset and there is **no way to visualize** the dataset, it is a good decision to use DBSCAN.

DBSCAN

- For an empirical understanding of DBSCAN, see the notebook "***DBSCAN: Density Based Clustering and Anomaly Detection***":
- https://github.com/rhasanbd/DBSCAN-Density-Based-Clustering-Anomaly-Detection/tree/master