Clustering - VIII

Prof. Dan A. Simovici

UMB

Density-based clustering was created to deal with the difficulties of model-based clustering. This type of clustering starts with the fundamental assumption that the set of objects to be clustered results from a mixture of Gaussian distributions.
Clusters are regarded as subsets of a dissimilarity space that have a high object density; these regions are separated by regions of low object density.
The first and the best known algorithm is `dbscan`. This was followed by `denclue` and `optics`.

`dbscan` is the best known density-based clustering algorithm.
The main idea of `dbscan` is the notion that objects are assigned to the same cluster if they are *density-reachable* from each other, a notion that we discuss next.

Definition

Let D be a subset of a dissimilarity space (S, d) referred to as *set of objects*, $D \subseteq S$, and let $B[x, \epsilon]$ be a closed sphere of radius ϵ centered in x, where $x \in D$. The set $B[x, \epsilon] \cap D$ is denoted by $B_D[x, \epsilon]$.
A *region of high density* around an object x is a a closed sphere $B[x, \epsilon]$ that contains at least μ objects of D.

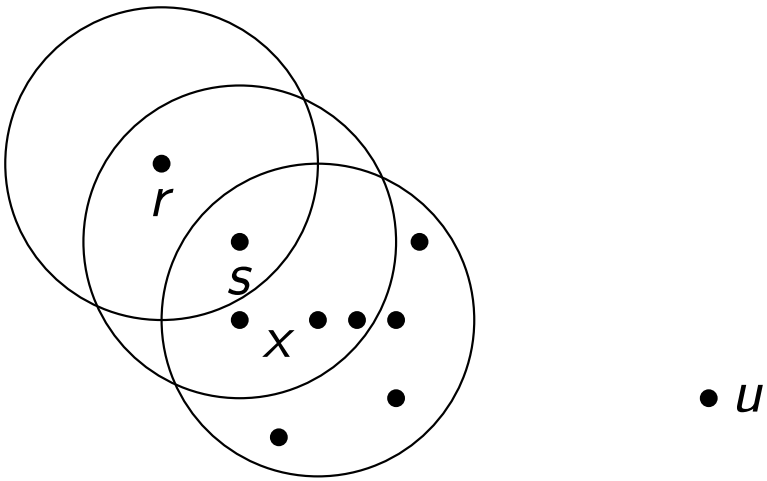
Definition

An object x is said to be an (ϵ, μ)-*core object* if $|B_D[x, \epsilon]| \geqslant \mu$.
An (ϵ, μ)-*border object* is an object y in $B_D[x, \epsilon]$, where x is a core object such that $|B_D[y, \epsilon]| < \mu$.
An object that is neither an (ϵ, μ)-core object nor an (ϵ, μ)-border object is an (ϵ, μ)-*noise object*.

To simplify the language, when ϵ and μ are fixed we omit references to these numbers and we use the terms core object, border object and noise object.

Core, border, and noise objects

Example



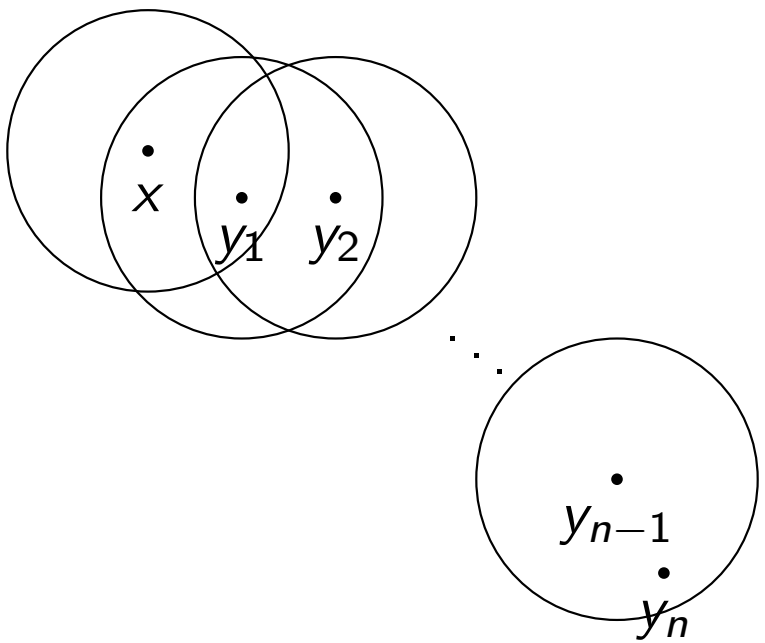
We have $|B_D[x, 2]| = 8$, $|B_D[s, 2]| = 5$, and $|B_D[r, 2]| = 2$. If $\mu = 6$ and $\epsilon = 2$, then x is a core object, s is a border object, and both r and u are noise objects.

Definition

An object y is *(ϵ, μ)-directly density-reachable from an (ϵ, μ)-core object object x* if $y \in B_D[x, \epsilon]$.

Definition

An object z is (ϵ, μ)-*density reachable from an object* x if there exists a sequence of objects $(y_1, \ldots, y_n)$ with $y_1 = x$ and $y_n = z$ such that y_{i+1} is (ϵ, μ)-directly density-reachable from y_i for $1 \leqslant i \leqslant n - 1$.



Definition

An object p is (ϵ, μ)-*density connected* to an object q if there exists a core object w such that both p and q are (ϵ, μ)-density reachable from w.

Note that:

- if z is (ϵ, μ)-density reachable from x through a chain of objects, then the intermediary objects $y_1, \ldots, y_{n-1}$ must be core points;
- if z is (ϵ, μ)-density reachable from a core point x and u is (ϵ, μ)-density reachable from a core point z, then u is (ϵ, μ)-density reachable from x. In other words, the density reachability is a transitive relation.

Thus, the density-reachability relation is the transitive closure of the direct density-reachability. However, this relation is not symmetric although its restriction to core objects is symmetric.
Density connectivity is a symmetric relation. Its restriction to density reachable points is reflexive. Also, it is clear that if an object p is density recheable from a core object o, then p is density connected to o.

Now we can define the notion of cluster as it is used in the `dbscan` algorithm.

Definition

Let D be a collection of objects. An (ϵ, μ)-*cluster* is a non-empty subset C of D that satisfies the following conditions:

1. C contains at least one (ϵ, μ)-core point;
2. for all $u, v \in D$, if $u \in C$ and v is (ϵ, μ)-density reachable from u (which implies that u is a core object), then $v \in C$ (the maximality property);
3. for all $u, v \in C$, u is (ϵ, μ)-density connected to v.

Two border objects of the same cluster C are not necessarily density-reachable from each other. However, there must be a core object in C from which both border objects are density-reachable.

Definition

Let $C_1, \ldots, C_k$ be the (ϵ, μ)-clusters of a database D. The *noise* is the set of objects in D that do not belong to any cluster C_j.

Lemma

Let D be a collection of objects and let $C_1, \ldots, C_k$ be the (ϵ, μ)-clusters of D. The following hold:

1. *each cluster C_j contains at least μ points;*
2. *if $p \in D$ and $|B_D[p, \epsilon]| \geqslant \mu$, then the set O defined as*

$$O = \{o \in D \mid o \text{ is } (\epsilon, \mu)\text{-density reachable from } p\}$$

   is a (ϵ, μ)-cluster;
3. *if C is a (ϵ, μ)-cluster and $w \in C$ is a (ϵ, μ)-core object in C then*

$$C = \{o \in D \mid o \text{ is } (\epsilon, \mu)\text{-density reachable from } w\}.$$

Let C_j be an (ϵ, μ)-cluster. Since C_j contains at least an object p, p must be density-connected with itself via some object o in C_j. Thus, at least o must satisfy the core object condition and, therefore $|B_D(o, \epsilon)| \geqslant \mu$, which proves Part (1).
To prove Part (2) let

$$O = \{o \in D \mid o \text{ is } (\epsilon, \mu)\text{-density reachable from } p\}.$$

Suppose that $u \in O$ and v is (ϵ, μ)-density reachable from u. Since $u \in O$, it follows that u is (ϵ, μ)-density reachable from p; therefore, $u \in O$, and the first property of Definition given on Slide 13 is satisfied.

Let now $u, v \in O$. In this case, both u and v are (ϵ, μ)-density reachable from p, which means that they are (ϵ, μ)-density connected. Thus, O is indeed a cluster.
Let C be a cluster and suppose that o is an object such that o is density-reachable from a core object w of C. Then, $o \in C$ by the maximality property of clusters.
Conversely, suppose that o belongs to C. Note that C must contain a core point v, hence o is density connected to the point v by the definition of clusters. Thus, there exists a core object $p \in C$ such that both o and v are (ϵ, μ)-density reachable from p. This implies that o is density reachable from p.
This concludes the third part of the proof.

Density connectivity is a symmetric relation. Its restriction to density reachable points is reflexive. Also, it is clear that if an object p is density recheable from a core object o, then p is density connected to o.

- `dbscan` starts with an arbitrary point p and retrieves all points density-reachable from p with respect to the parameters ϵ and μ.
- If p is a core point, this procedure yields a cluster relative to ϵ and μ.
- If p is a border point, no points are density-reachable from p and `dbscan` visits the next point of the database.

- Since we use global values for ϵ and μ, `dbscan` may merge two clusters according to Definition on Slide 13 into one cluster, if two clusters of different density are "close" to each other.
- Two sets of points having at least the density of the thinnest cluster will be separated from each other only if the distance between the two sets is larger than ϵ. Consequently, a recursive call of `dbscan` may be necessary for the detected clusters with a higher value for μ. This is, however, no disadvantage because the recursive application of `dbscan` yields an elegant and very efficient basic algorithm.
- Furthermore, the recursive clustering of the points of a cluster is only necessary under conditions that can be easily detected.

An informal description of `dbscan` is given next. The algorithm makes use of the function `ExpandCluster` described subsequently.

- The set `SetOfObjects` represents either the whole set of objects or the discovered cluster from the previous run.
- The variable `ClusterId` ranges over a countable data type whose first value is `unclassified`, second value is `noise` followed by other values which are integers:

$$\texttt{unclassified} < \texttt{noise} < 1 < 2 < \cdots$$

- Each object is marked with a `ClusterId` value, `Object.ClId`.
- The function `nextId(ClusterId)` returns the successor of `clusterId`. The function `SetOfObjects`.*get*(i) returns the i^{th} element of `SetOfObjects`.

These assumptions and notations are used in the description of the `dbscan` algorithm.
Clusters are discovered in a two-step approach:

- choose an arbitrary point in the data set satisfying the core point condition as a *seed*;
- retrieve all points that are density-reachable from the seed, this obtaining the cluster that contains the seed.

A cluster C contains exactly the points that are density-reachable from an arbitrary core point of C.

Algorithm 1: The `dbscan` Algorithm

Data: A set of points `SetOfObjects`, ϵ and μ
Result: A density-based clustering of `SetOfPoints`

```
1 Initially objects in SetOfObjects are unclassified;
2 ClusterId := nextId(noise);
3 for i ← 1 to |SetOfObjects| do
4     Object ← SetOfObjects.get(i) ;
5     if Object.ClId is unclassified then
6         if ExpandCluster(SetOfObjects,Object,ClusterId,ϵ,μ) then
7             ClusterId ← nextId(ClusterId);
```

The function `ExpandCluster` is given below. In this function, the function `SetOfPoints.regionQuery`(p, ϵ) is called in order to compute $B_D[p, \epsilon]$ as a list of points.

```
1  ExpandCluster(SetOfPoints,Point,ClId,ε,μ):Boolean{ seeds ←
     SetOfPoints.regionQuery(Point,ε);
2  if seeds.size < μ then
3      SetOfPoints.changeClId(Point,NOISE);
4      return false;
5  else
6      setOfPoint.changeClId(seeds,ClId);
7      seeds.delete(Point);
8      while seeds ≠ EMPTY do
9          currentP ← seeds.first;
10         result ← SetOfPoints.regionQuery(currentP,ε);
11         if result.size ⩾ μ then
12             for i ← 1 to result.size do
13                 resultP ← result.get(i);
14                 if resultP.ClId ∈ {unclassified, noise} then
15                     if resultP.ClId == unclassified then
16                         seeds.append(resultP);
17                 SetOfPoints.changeClId(resultP.ClId);
```

`SetOf Points` is either the whole database or a discovered cluster from a previous run while ϵ and μ are the global density parameters determined either manually or according to a certain heuristics.
The function `SetOf Points.get(i)` returns the i^{th} element of `SetOfPoints`. The most important function used by `dbscan` is the function `ExpandCluster` shown next.

The density-based algorithm in `dbscan` relies on computing all points belonging to an ϵ-neighborhood.
A simple approach is to perform a linear search, i.e., always calculating the distances to all other points to find the closest points. This requires $O(n)$ operations, with n being the number of data points, for each time a neighborhood is needed. Since `dbscan` deals with each data point once, this results in a $O(n^2)$ runtime complexity.

A convenient way in **R** is to compute a distance matrix with all pairwise distances between points and sort the distances for each point (row in the distance matrix) to precompute the nearest neighbors for each point. However, this method has the drawback that the size of the full distance matrix is $O(n^2)$, and becomes very large and slow to compute for medium to large data sets.

To avoid computing the complete distance matrix, `dbscan` relies on kd-trees.

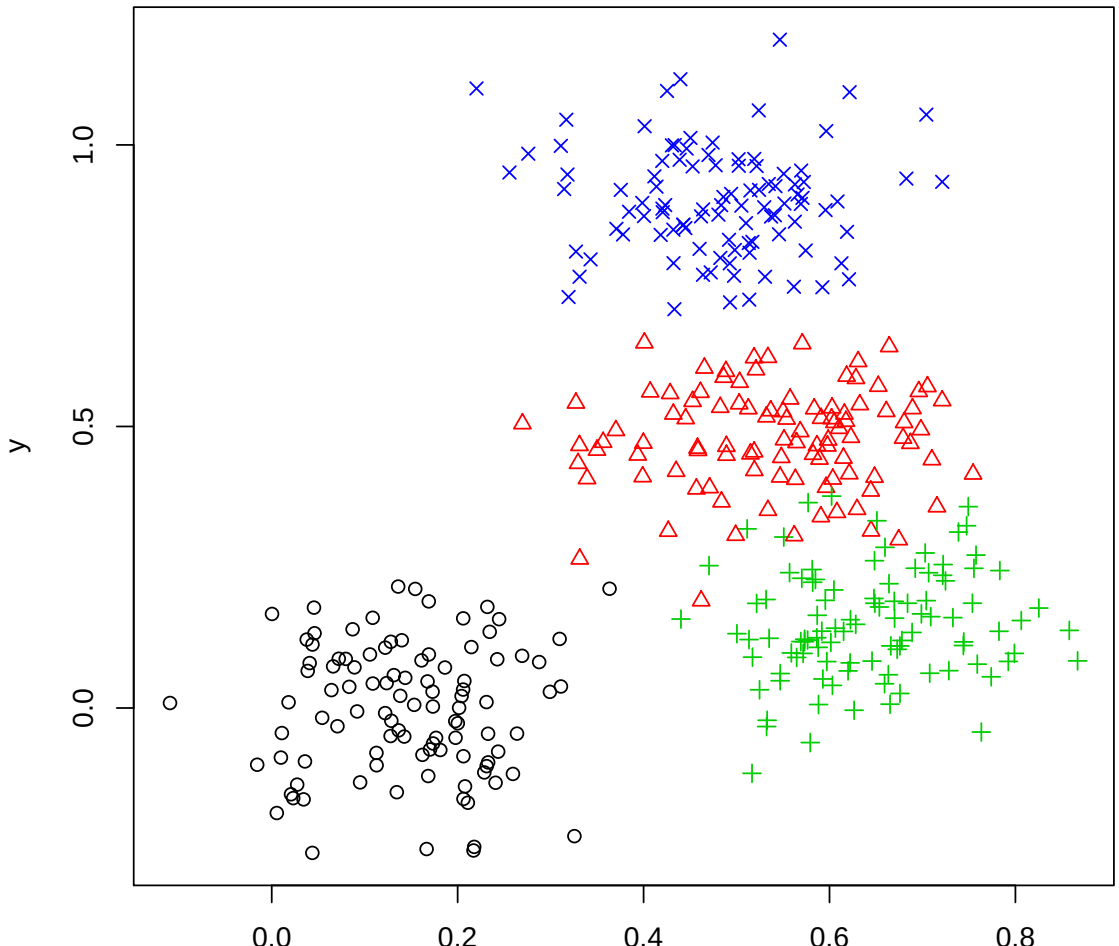
This data structure allows `dbscan` to identify all neighbors within a fixed radius ϵ more efficiently in sub-linear time using on average only $O(n \log n)$ operations per query. This results in a reduced runtime complexity of $O(n \log n)$.

A bidimensional artificial data set that consists of four Gaussian clusters with 100 points each is produced using the following piece of code:

```
> set.seed(665544)
> n <- 400
> z <- cbind(
+ x = runif(4,0,1) + rnorm(n,sd = 0.1),
+ y = runif(4,0,1) + rnorm(n,sd = 0.1)
+ )
> true_clusters <- rep(1:4,time=100)
```

The data set is produced with

```
> pdf("cldbscan.pdf")
> plot(z,col= true_clusters,pch=true_clusters)
> dev.off()
```

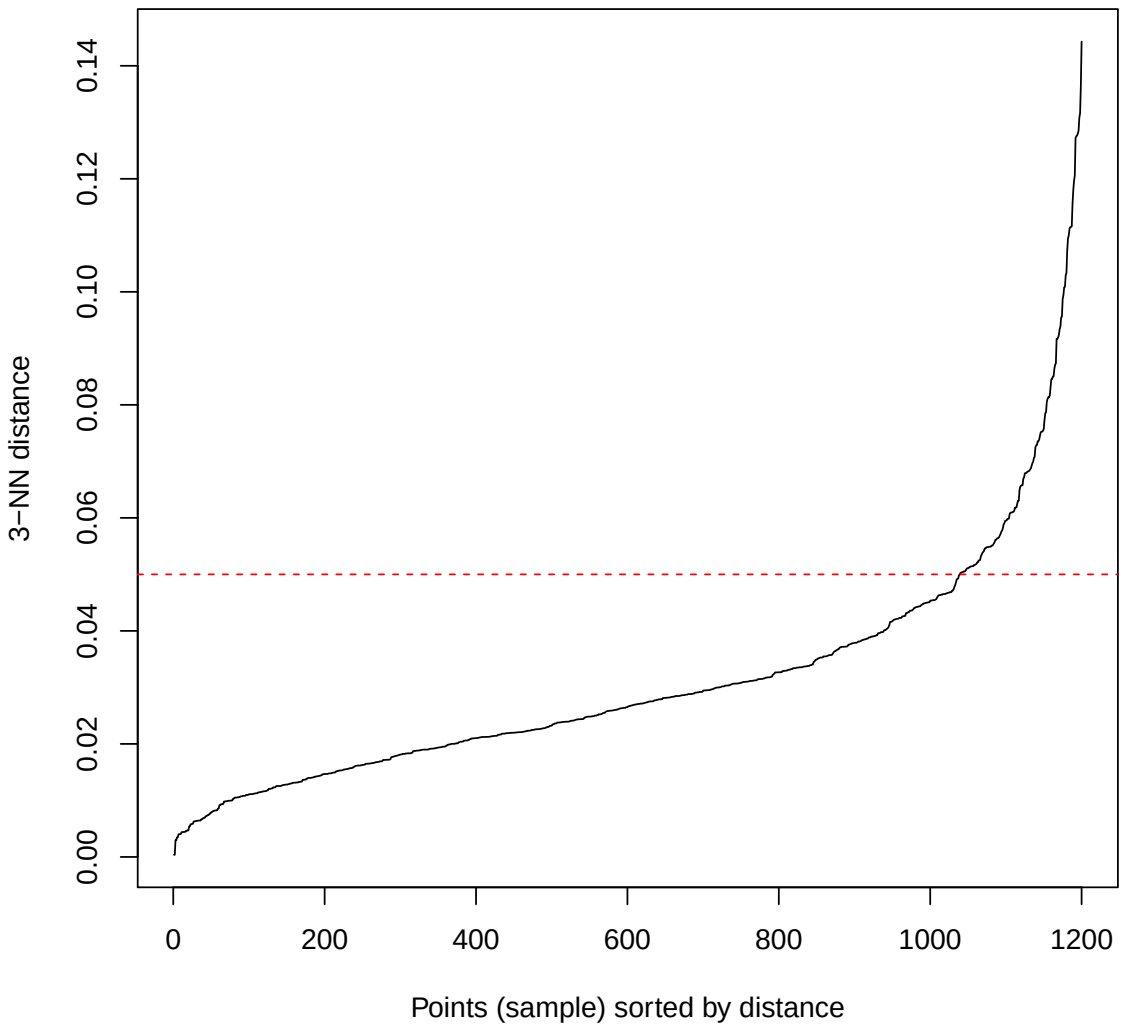


A practical choice for μ (denoted here by `minPts`) is the number of dimensions plus 1. In this case, the chosen μ is 3.
To decide on the neighborhood radius ϵ one can use the function `kNNdisplot` computed by

```
> kNNdistplot(z,k=3)
> abline(h=.05, col="red", lty=2)
```

This function yields a plot of the distances to the k^{th} nearest neighbor in decreasing order. The idea is that points located inside clusters will have small k-nearest neighbor distance because of their proximity to other points in the same cluster, while noise points are isolated and have a rather large kNN distance.

The resulting graph is shown below. We look for a knee in the plot and one can be found at $\epsilon = 0.05$; this is shown in the graph by a horizontal line at 0.05.



Thus, we can apply dbscan with the parameters $\epsilon = 0.05$ and $\mu = 3$ by writing:

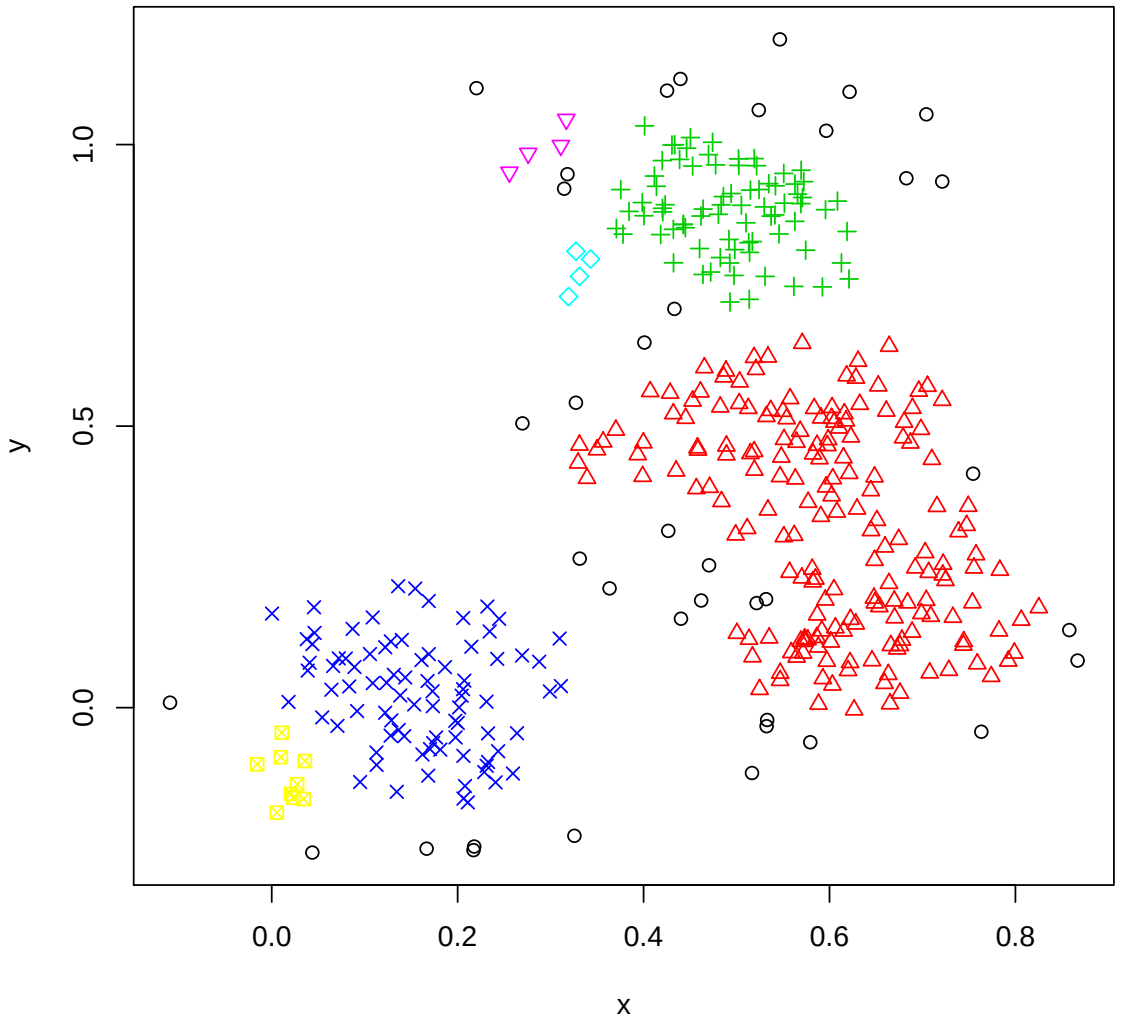
```
> res <- dbscan(z, eps=0.05, minPts = 3)

> res
dbscan clustering for 400 objects.
Parameters: eps = 0.05, minPts = 3
The clustering contains 6 cluster(s) and 38 noise points.

  0   1   2   3   4   5   6
 38 182  79  84   4   4   9

Available fields: cluster, eps, minPts
```

```
> predict(res,z[1:25,],data = z)
 [1] 0 1 1 2 3 1 1 4 6 1 1 2 3 1 1 0 0 1 1 2 3 1 1 2 0
```

The final scatter plot is obtained with

```
> plot(z,col=res$cluster + 1L,pch=res$cluster + 1L)
```

and is shown in the next slide.


y
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
x

This scatter plot shows that `dbscan` correctly identfies the upper cluster and the cluster located in right lower quadrant but merged the two remaining clusters because the region beteen them has a sufficiently high density. The remaining small clusters are isolated groups of 3 points (passing minPts) and the isolated noise points.

Finally, the function `hullplot` adds convex closures of the clusters to the previous scatter plots. When applied as

```
> hullplot(z,res)
```

the results are shown below: