DBSCAN: quick overview

Rei Sanchez-Arias, Ph.D.

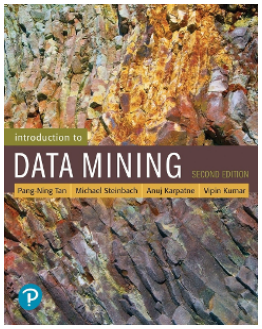
Introducing the Density-Based Spatial Clustering and Application with Noise (DBSCAN) method

Some Resources

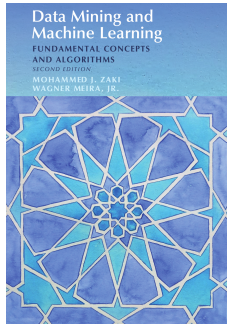
Some resources and motivation

- "Visualizing DBSCAN Clustering"
- DBSCAN original paper from KDD-96
- DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN
- `dbscan` R package
- `sklearn.cluster.DBSCAN`

"Introduction to Data Mining" by Tan, Steinbach, Karpatne, Kumar



"Data Mining and Machine Learning: Concepts and Algorithms" by Zaki and Meira



> Examples and materials in this set of slides are adapted from the books mentioned above.

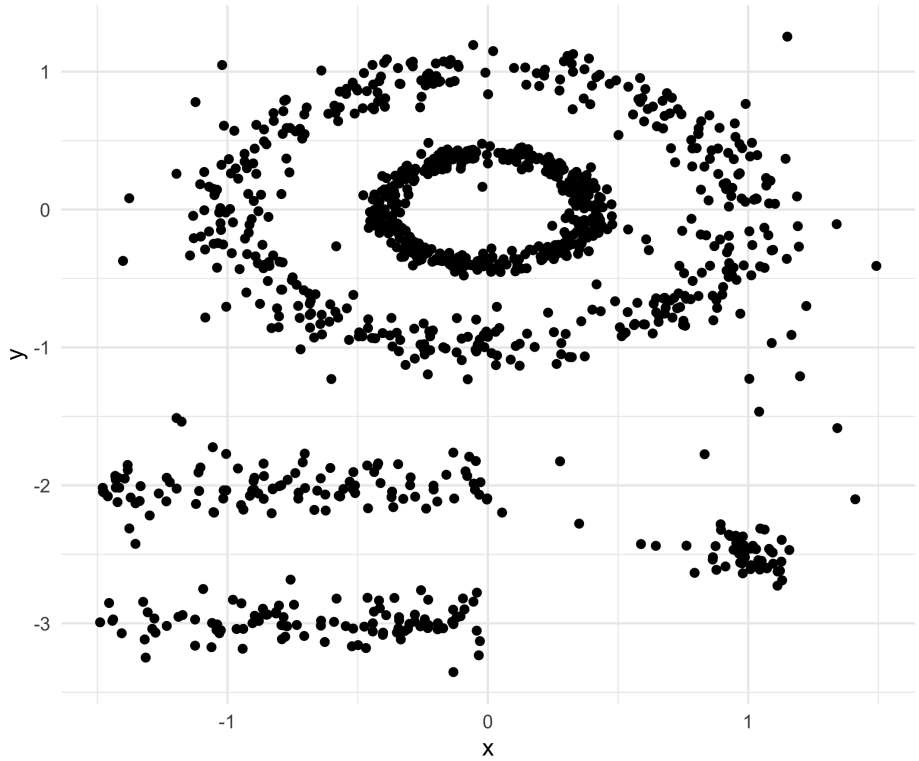
Clustering

```
library(tidyverse)
library(factoextra)
```

Consider a visualization of the dataset `multishapes` (part of the `factoextra` package)

```
ggplot(data = multishapes) +
  geom_point(aes(x = x, y = y)) +
  theme_minimal()
```

How many groups do you observe?



Try K-means

Try first the k-means method to find groups in the dataset `multishapes`. Check first the structure of the object `multishapes`

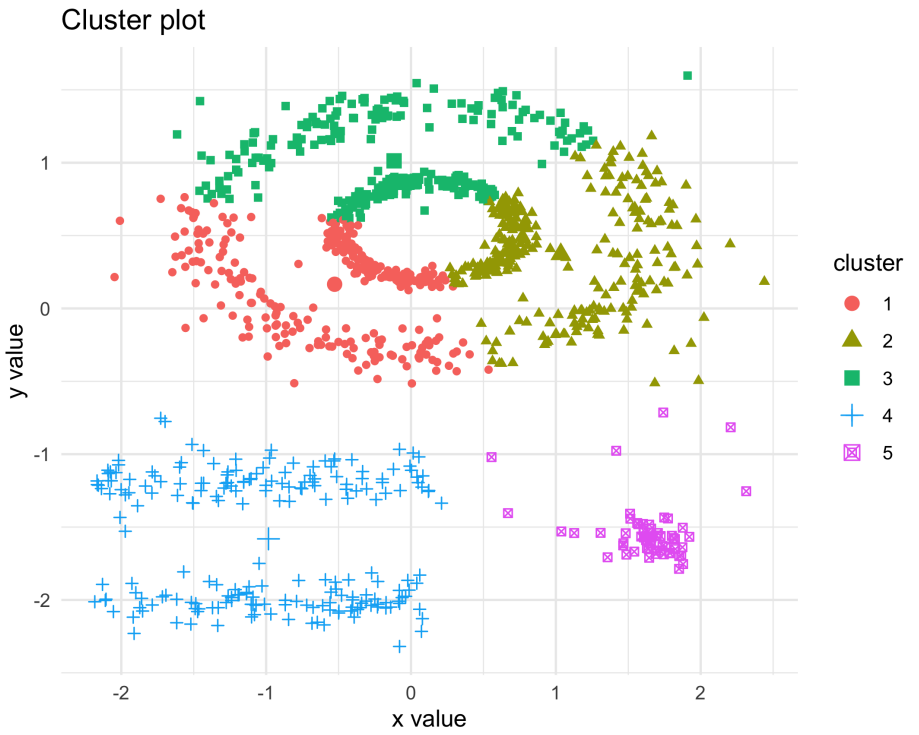
```
str(multishapes)
```

```
## 'data.frame':    1100 obs. of  3 variables:
##  $ x    : num  -0.804 0.853 0.927 -0.753 0.707
##  $ y    : num  -0.853 0.368 -0.275 -0.512 0.81
##  $ shape: num  1 1 1 1 1 1 1 1 1 1 ...
```

```
# use columns x and y from multishapes
df_multi <- multishapes[ , 1:2]
multi_km <- kmeans(df_multi, centers = 5)
```

Did you expect these groups?

```
fviz_cluster(multi_km, df_multi,
             geom = "point", ellipse = FALSE) +
  theme_minimal()
```



Introducing DBSCAN

DBSCAN

The Density-Based Spatial Clustering and Application with Noise method, in short DBSCAN, is a **density-based clustering algorithm** which can be used to identify clusters of multiple shapes, even when data contains *outliers* and *noise*.

> Check the original article by Ester et al.

One of the main observations presented in their work, is the fact that the **density of points** in a cluster is considerably higher than the density of points outside the cluster (area of noise). Also, for each point of a cluster, the **neighborhood** of a given radius (`eps`) has to contain at least a minimum number of points.

The goal is to identify dense regions, which can be measured by the number of objects close to a given point.

Density ~ number of points within a specified radius (`eps`)

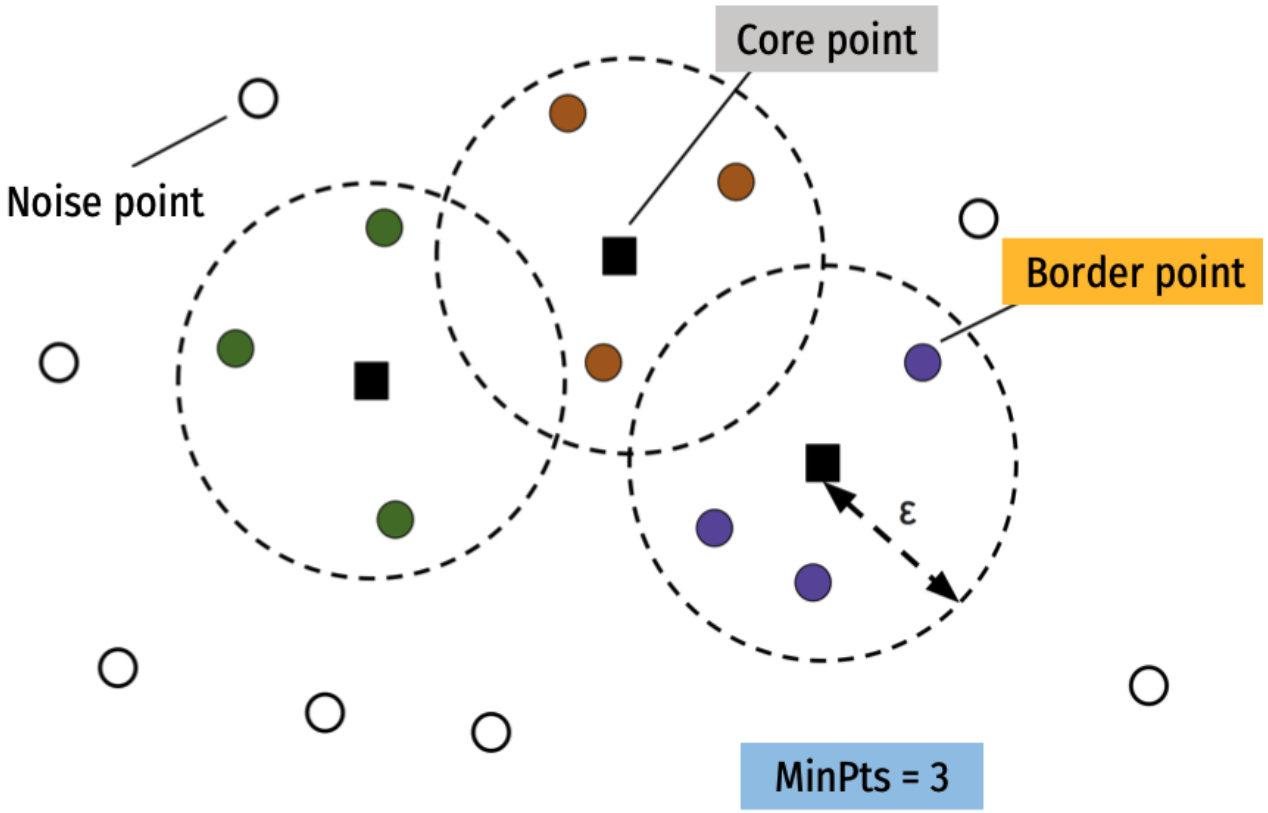
Some definitions

Two **important parameters** are required for DBSCAN: epsilon (`eps`) and minimum points (`MinPts`). The parameter `eps` defines the radius of neighborhood around a point `x`. It is called called the ϵ-neighborhood of x. The parameter `MinPts` is the minimum number of neighbors within ϵ radius.

- Any point `x` in the dataset, with a neighbor count greater than or equal to `MinPts`, is marked as a **core point**.
- We say that `x` is a **border point**, if the number of its neighbors is less than `MinPts`, but it belongs to the ϵ-neighborhood of some core point `z`.
- Finally, if a point is neither a core nor a border point, then it is called a **noise point** or an **outlier**.

Core, border, and noise points

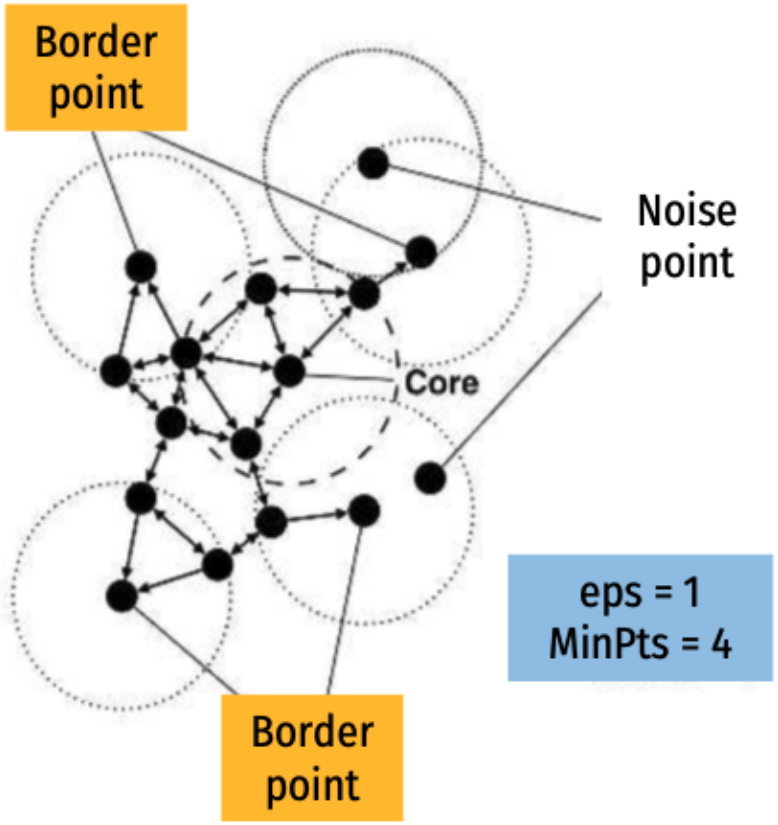
- **Core** point: has at least a specified number of points (`MinPts`) within `eps`. These are points that are at the interior of a cluster (counts the point itself)
- **Border** point: not a core point, but in the neighborhood of a core point
- **Noise** point: not a core point nor a border point



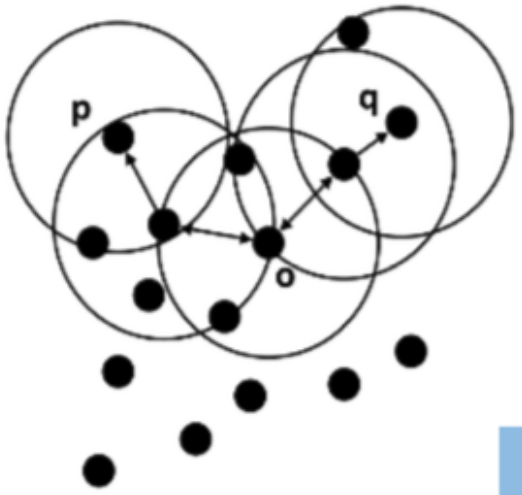
About neighbors

- **Direct density reachable**: A point A is directly density reachable from another point B if:
  - A is in the ϵ-neighborhood of B and
  - B is a core point.
- **Density reachable**: A point A is density reachable from B if there are a set of core points leading from B to A.
- **Density connected**: Two points A and B are density connected if there is a core point C, such that both A and B are density reachable from C.
- **Density-based cluster**: a group of density connected points.

About neighbors (cont.)



eps = 1
MinPts = 4



p and q are **density reachable** from o

Therefore p and q are **density-connected**

eps = 1
MinPts = 4

Algorithm

1. Label all points as core, border, or noise points.
2. Eliminate noise points.
3. Put an edge between all core points within a distance `eps` of each other.
4. Make each group of connected core points into a separate cluster.
5. Assign each border point to one of the clusters of its associated core points.

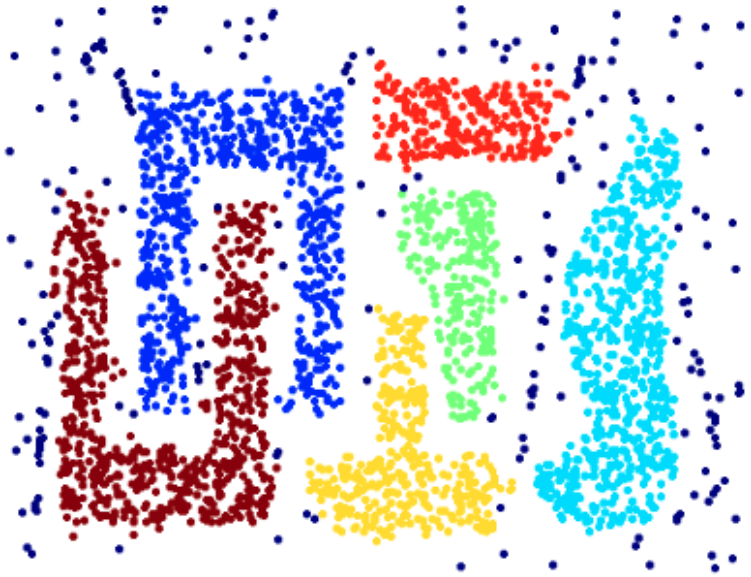
Advantages and Limitations

- DBSCAN does not require the user to pre-specify the number of clusters to be generated.
- DBSCAN can find any shape of clusters. (clusters do not have to be spherical)
- DBSCAN can identify outliers.

One limitation of DBSCAN is that it is sensitive to the choice of ϵ, in particular if clusters have different densities. If ϵ is too small, sparser clusters will be defined as noise. If ϵ is too large, denser clusters may be merged together. This implies that, if there are clusters with different local densities, then a single ϵ value may not suffice.

When DBSCAN works well

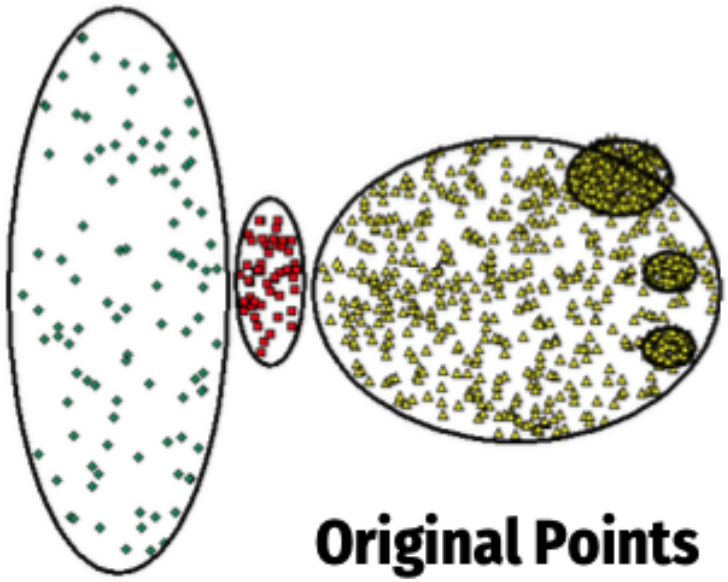
Original Points



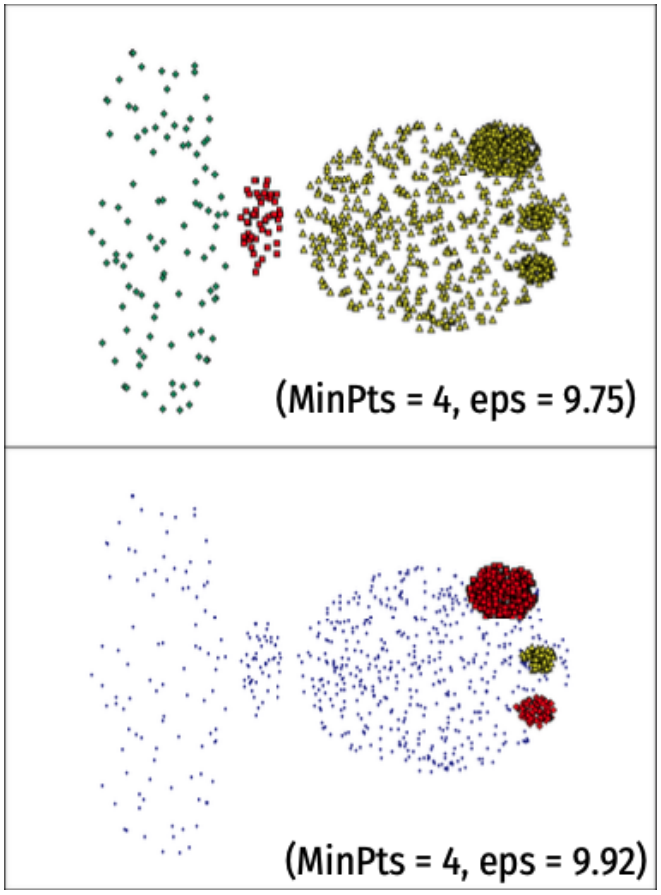
Clusters

- Resistant to Noise
- Can handle clusters of different shapes and sizes

When DBSCAN does NOT work well



- Varying densities
- High-dimensional data



`dbscan` R package

Implementation

We use the DBSCAN implementation from the `dbscan` package (a significantly faster and large-scale implementation of DBSCAN compared to other implementations such as the one in the `fpc` package)

```
library(dbscan)
multi_dbscan <- dbscan(df_multi, eps = 0.15, minPts = 5)
```

The result of the function `dbscan()` is an integer vector with cluster assignments (where `0` indicates noise points)

```
str(multi_dbscan)
```

```
## List of 3
##  $ cluster: int [1:1100] 1 1 1 1 1 1 1 1 1 1 ...
##  $ eps    : num 0.15
##  $ minPts : num 5
##  - attr(*, "class")= chr [1:2] "dbscan_fast" "dbscan"
```

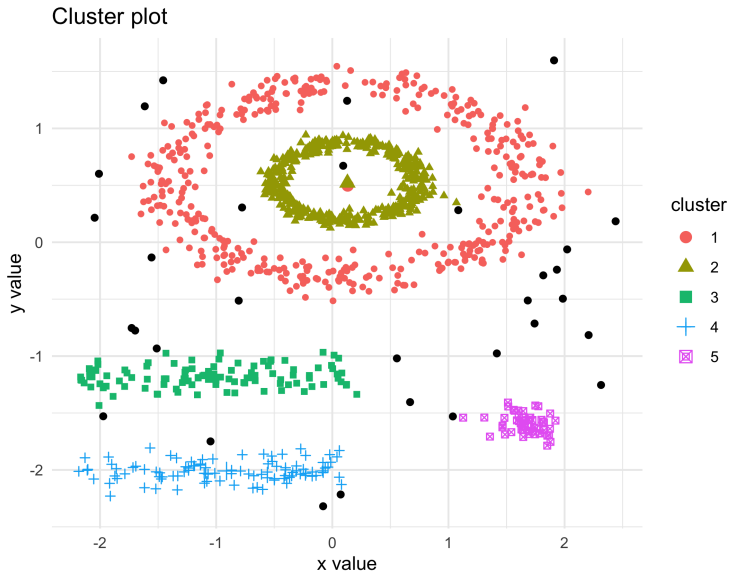
DBSCAN results

We can now plot the `multishapes` data again coloring every point based on the cluster assignment found by DBSCAN

```
# print the results
multi_dbscan
```

```
## DBSCAN clustering for 1100 objects.
## Parameters: eps = 0.15, minPts = 5
## The clustering contains 5 cluster(s) and 31 nc
##
##   0   1   2   3   4   5
##  31 410 405 104  99  51
##
## Available fields: cluster, eps, minPts
```

```
fviz_cluster(multi_dbscan, df_multi,
             geom = "point", ellipse = F) +
  theme_minimal()
```



Black points are outliers here. Dataset contains 5 clusters.

Choosing `eps` and `minPts`

- The method proposed here consists of computing the **k-nearest neighbor distances** in a matrix of points. The idea is to calculate, the average of the distances of every point to its `k` nearest neighbors. The value of `k` will be specified by the user and corresponds to `minPts`.
- Next, these k-distances are plotted in an ascending order. The aim is to **determine the "knee/bend"**, which corresponds to the optimal `eps` parameter. A knee corresponds to a threshold where a **sharp change** occurs along the k-distance curve.
- A rule of thumb for `minPts` is to use *at least* the number of dimensions of the data set plus one. For `eps`, we can plot the points' kNN distances (i.e., the distance to the kth nearest neighbor) in decreasing order and look for a knee in the plot.

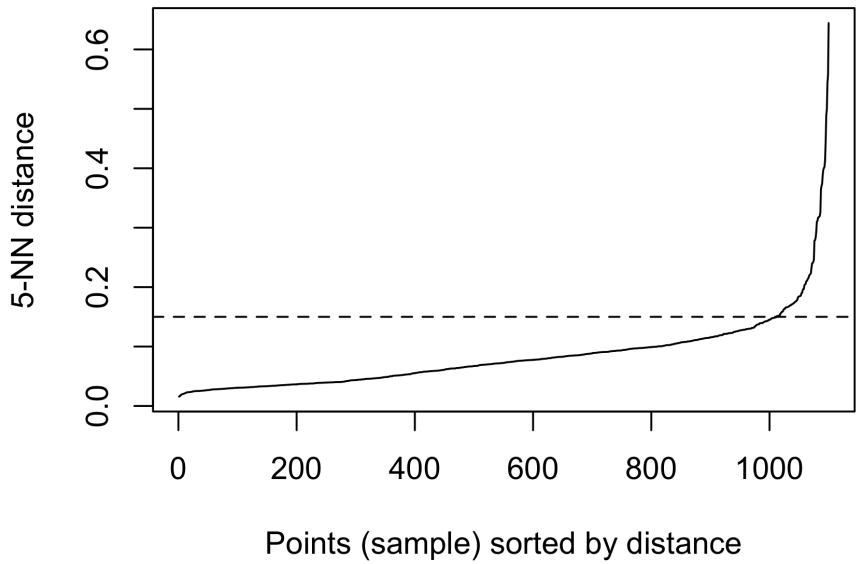
Choosing `eps` and `minPts` (cont.)

The idea behind this *heuristic* is that points located inside of clusters will have a small k-nearest neighbor distance, because they are close to other points in the same cluster, while noise points are isolated and will have a rather large kNN distance.

`dbscan::kNNdistplot()` is used for the k-distance plot:

```
kNNdistplot(df_multi, k = 5)
# draw a dashed line where the bend happens
# around a distance of 0.15
abline(h = 0.15, lty = 2)
```

Other heuristics are to choose `minPts` using $\ln(n)$, where n is the total number of points to be clustered, or set its value to $2m$ where m is the dataset dimensionality (check here)



Cluster validation

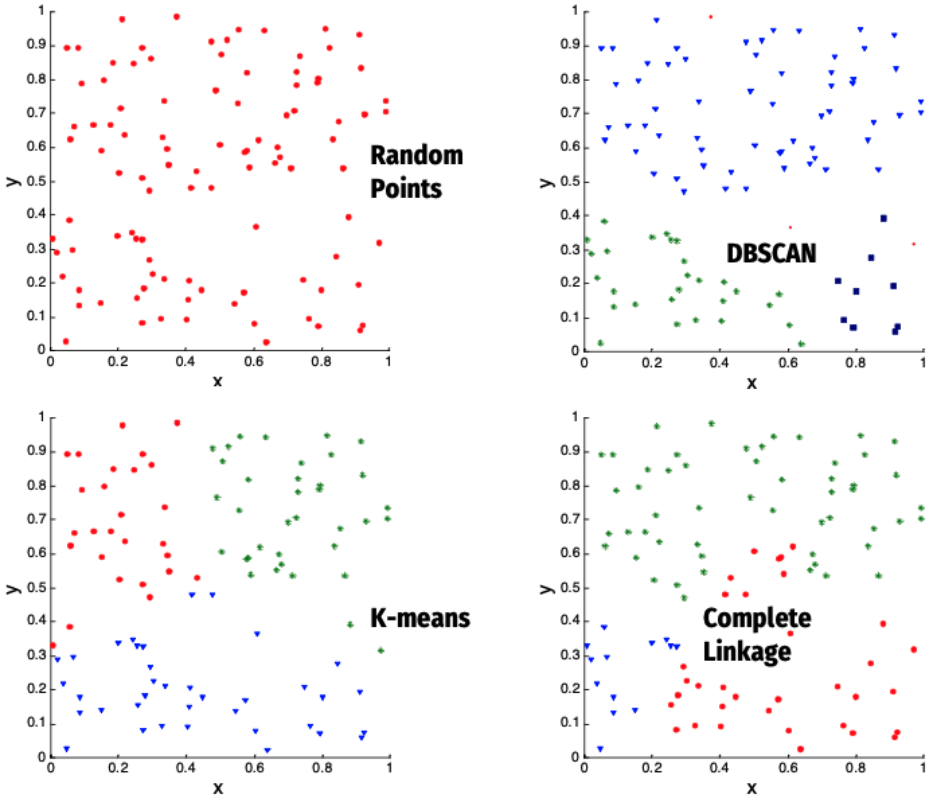
See Chapter 17 of Zaki and Meira

Cluster validity

How to evaluate the **"goodness" of the resulting clusters**?

We evaluate them:

- to **avoid finding patterns in noise**,
- to **compare clustering** algorithms,
- to **compare two sets** of clusters,
- to **compare two clusters**



Aspects of cluster validation

1. Determining the clustering tendency of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
2. Comparing the results of a cluster analysis to externally known results, e.g., to externally given class labels.
3. Evaluating how well the results of a cluster analysis fit the data without reference to external information. Use only the data
4. Comparing the results of two different sets of cluster analyses to determine which is better.
5. Determining the ‘correct’ number of clusters.

For 2, 3, and 4 we can further distinguish whether we want to evaluate the entire clustering or just individual clusters.

Some measures of cluster validity

Numerical measures that are applied to judge various aspects of cluster validity, can be classified into the following types: (check notes in Chapter 17 of Zaki and Meira)

- **External Index:** Used to measure the extent to which cluster labels match ***externally supplied*** class labels (or expert-specified knowledge). Some measures include: purity, maximum matching, F-measure, entropy, normalized mutual information, variation of information, Rand statistic.
- **Internal Index:** Used to measure the goodness of a ***clustering structure*** without respect to external information (derived from the data itself, such as SSE). Some measures include C-index, Dunn index, Silhouette coefficient, Hubert Statistic.
- **Relative Index:** Used to ***compare two different*** clusterings or clusters (for example comparing results obtained via different parameter settings for the same algorithm). Some measures include: gap statistic and silhouette coefficient.

Sometimes these are referred to as criteria instead of indices. However, sometimes criterion is the general strategy and index is the numerical measure that implements the criterion.

Cohesion and separation

Cohesion: measures how **closely related** are objects in a cluster. Sum of squared errors (SSE) can be used here:

$$WSS = \sum_i \sum_{x \in C_i} (x - m_i)^2$$

where m_i is the representative point for cluster C_i. Note:

- $WSS + BSS$ = constant.
- A larger number of clusters tend to result in smaller SSE

Separation: measures how **distinct** or well-separated a cluster is from other clusters. We can measure it using the between cluster sum of squares

$$BSS = \sum_i |C_i|(m - m_i)^2$$

where m is the centroid of the whole data set, $|C_i|$ is the size of the cluster C_i