Clustering algorithms

K-means and DBSCAN

Hicham Zmarrou, PhD

2018-11-22

Aims of this lesson

- Introduce you to the clustering algorithms and evaluation

Aims of this lesson

- Introduce you to the clustering algorithms and evaluation
- Give some application examples

Introduction

- unsupervised ML algorithms. Why unsupervised ?

Introduction

- unsupervised ML algorithms. Why unsupervised ?
- The model is trained based on given input variables which attempt to discover intrinsic groups (or clusters).

Introduction

- unsupervised ML algorithms. Why unsupervised ?
- The model is trained based on given input variables which attempt to discover intrinsic groups (or clusters).
- Clustering algorithms are widely used across all industries such as retail, banking, manufacturing, healthcare, etc.

Introduction

- unsupervised ML algorithms. Why unsupervised ?
- The model is trained based on given input variables which attempt to discover intrinsic groups (or clusters).
- Clustering algorithms are widely used across all industries such as retail, banking, manufacturing, healthcare, etc.
- Separate customers sharing similar characteristics.

Introduction

- unsupervised ML algorithms. Why unsupervised ?
- The model is trained based on given input variables which attempt to discover intrinsic groups (or clusters).
- Clustering algorithms are widely used across all industries such as retail, banking, manufacturing, healthcare, etc.
- Separate customers sharing similar characteristics.
- Extract abnormal observations

Types of Clustering Techniques

- Many types of clustering algorithms exist. Among these different clustering algorithms, there exists clustering behaviors known as

Types of Clustering Techniques

- Many types of clustering algorithms exist. Among these different clustering algorithms, there exists clustering behaviors known as
- Model based clustering: In this technique, the probability or likelihood of an observation being partitioned into a cluster is calculated.

Types of Clustering Techniques

- Many types of clustering algorithms exist. Among these different clustering algorithms, there exists clustering behaviors known as
- Model based clustering: In this technique, the probability or likelihood of an observation being partitioned into a cluster is calculated.
- Non-parametric clustering: an observation is partitioned into exactly one cluster (no probability is calculated).

Distance calculation for clustering

1. Distance calculations for quantitative variables are highly influenced by variable units and magnitude. Standardize!

Suppose, we are given a 2-dimensional data with

$$X = (x_1, x_2, \cdots, x_p)$$

and

$$Y = (y_1, y_2, \cdots, y_p)$$

We can calculate various distances as follows:

Distance calculation for clustering

1. Distance calculations for quantitative variables are highly influenced by variable units and magnitude. Standardize!
2. Use of a particular distance measure depends on the variable types.

Suppose, we are given a 2-dimensional data with

$$X = (x_1, x_2, \cdots, x_p)$$

and

$$Y = (y_1, y_2, \cdots, y_p)$$

We can calculate various distances as follows:

Distance calculation for clustering

Numeric variables

- Euclidean Distance: $d(X, Y) = \sum(x_i - y_i)^2$

Catergorical variables

Hamming Distance: $d(x, y) = \sum(x_i \neq y_i)$

Mixed variables

Distance calculation for clustering

Numeric variables

- Euclidean Distance: $d(X, Y) = \sum(x_i - y_i)^2$
- Manhattan Distance: $d(X, Y) = \sum |x_i - y_i|$

Catergorical variables

Hamming Distance: $d(x, y) = \sum(x_i \neq y_i)$

Mixed variables

Distance calculation for clustering

Numeric variables

- Euclidean Distance: $d(X, Y) = \sum(x_i - y_i)^2$
- Manhattan Distance: $d(X, Y) = \sum |x_i - y_i|$

Catergorical variables

Hamming Distance: $d(x, y) = \sum(x_i \neq y_i)$

Mixed variables

- Gower Distance

Distance calculation for clustering

Numeric variables

- Euclidean Distance: $d(X, Y) = \sum(x_i - y_i)^2$
- Manhattan Distance: $d(X, Y) = \sum |x_i - y_i|$

Catergorical variables

Hamming Distance: $d(x, y) = \sum(x_i \neq y_i)$

Mixed variables

- Gower Distance
- Cosine Similarity

Distance calculation for clustering

1. Julie loves me more than Linda loves me

List of the words from both texts

me Julie loves Linda than more likes Jane

words	frequency 1	frequency 2
me	2	2
Jane	0	1
Jullie	1	0
Linda	0	1
likes	0	1
loves	2	1
more	1	1
than	1	1

Distance calculation for clustering

1. `Julie loves me more than Linda loves me`
2. `Jane likes me more than Julie loves me`

List of the words from both texts

`me Julie loves Linda than more likes Jane`

words	frequency 1	frequency 2
me	2	2
Jane	0	1
Jullie	1	0
Linda	0	1
likes	0	1
loves	2	1
more	1	1
than	1	1

Distance calculation for clustering

Cosine Similarity

The two vectors are, again:

```
a: [2, 1, 0, 2, 0, 1, 1, 1]

b: [2, 1, 1, 1, 1, 0, 1, 1]
```

$$\text{cos-similarity}(X, Y) = \cos(X, Y) = \frac{X.Y}{||X||.||Y||}$$

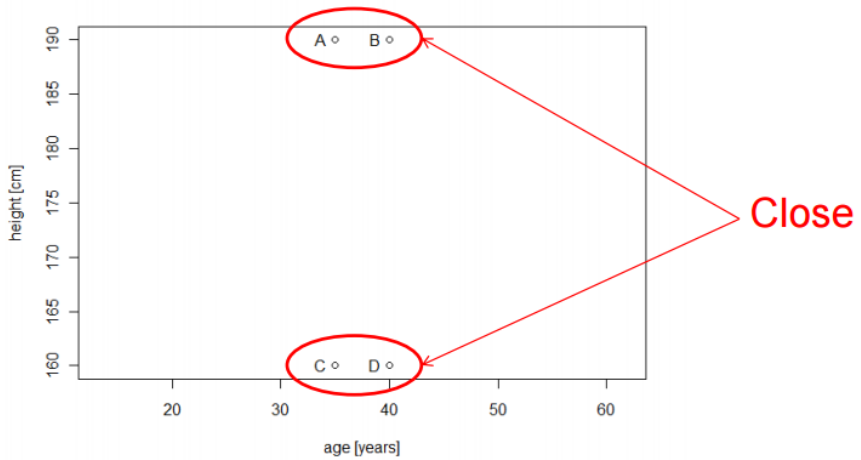
The cosine of the angle between a and b is about 0.822.

Why Standardization?

Person	Age [years]	Height [cm]
A	35	190
B	40	190
C	35	160
D	40	160

Example 1: cm

- 4 persons

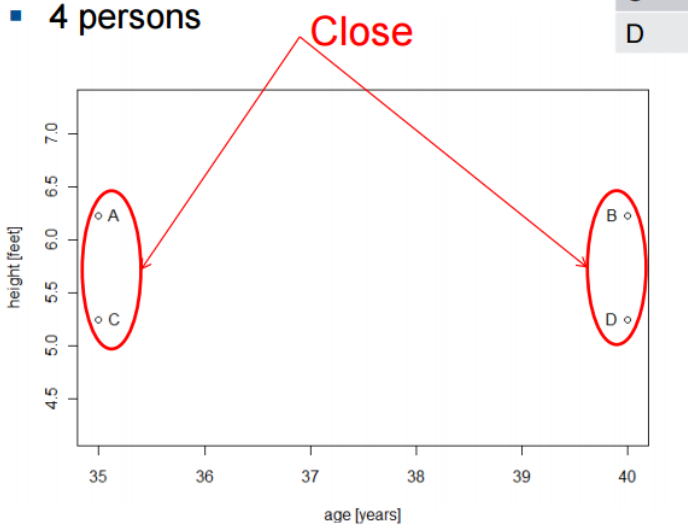


Why Standardization?

Person	Age [years]	Height [feet]
A	35	6.232
B	40	6.232
C	35	5.248
D	40	5.248

Example 1: feet

- 4 persons

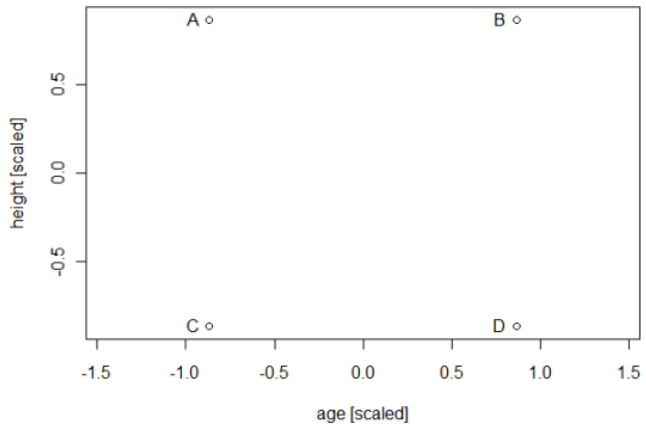


Why Standardization?

Example 1: scaled

- 4 persons

Person	Age [scaled]	Height [scaled]
A	-0.87	0.87
B	0.87	0.87
C	-0.87	-0.87
D	0.87	-0.87



No subgroups anymore

K-means clustering

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations
- Repeat until convergence:

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations
- Repeat until convergence:
  - for each point x_i:

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations
- Repeat until convergence:
  - for each point x_i:
    - find nearest centroid c_j

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations
- Repeat until convergence:
  - for each point x_i:
    - find nearest centroid c_j
    - assign the point x_i to cluster c_j

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations
- Repeat until convergence:
  - for each point x_i:
    - find nearest centroid c_j
    - assign the point x_i to cluster c_j
  - for each cluster $c_1, \cdots, c_k$

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations
- Repeat until convergence:
  - for each point x_i:
    - find nearest centroid c_j
    - assign the point x_i to cluster c_j
  - for each cluster $c_1, \cdots, c_k$
    - compute new centroid c_j = mean of all points x_i

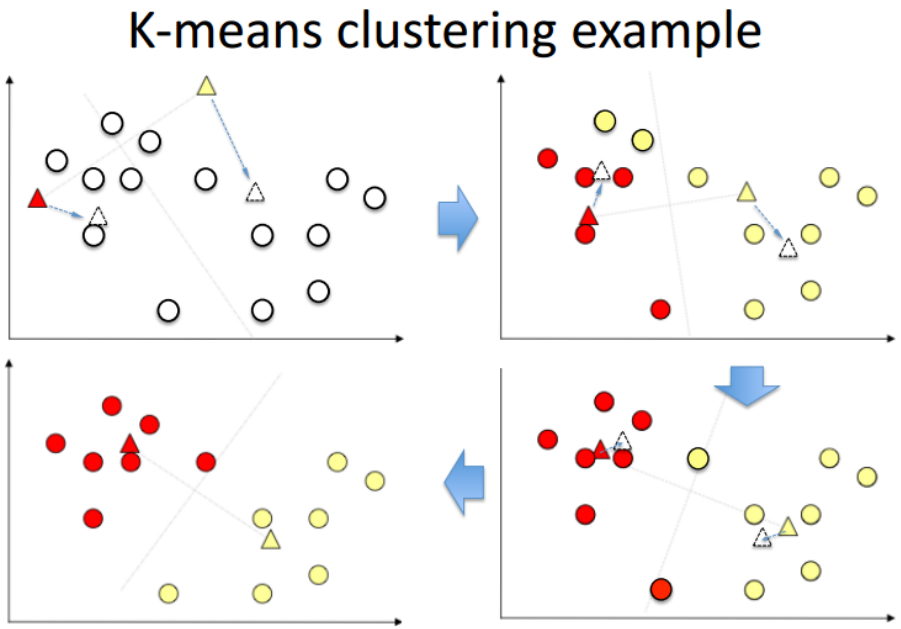
K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations
- Repeat until convergence:
  - for each point x_i:
    - find nearest centroid c_j
    - assign the point x_i to cluster c_j
  - for each cluster $c_1, \cdots, c_k$
    - compute new centroid c_j = mean of all points x_i
    - assign the new centroid to the cluster j in the previous step

K-means clustering: How does it work ?

- Input: K, set pf points $x_1, \cdots, x_n$
- Place random centroids $c_1, \cdots, c_k$ at random locations
- Repeat until convergence:
  - for each point x_i:
    - find nearest centroid c_j
    - assign the point x_i to cluster c_j
  - for each cluster $c_1, \cdots, c_k$
    - compute new centroid c_j = mean of all points x_i
    - assign the new centroid to the cluster j in the previous step
- Stop when none of the cluster assignments change

K-means clustering: How does it work ?



K-means clustering in R

Function `kmeans` from package `kmeans`

```
kmeans(x, centers, iter.max = 10, nstart = 1)
```

- *x*: numeric matrix, numeric data frame or a numeric vector

K-means clustering in R

Function `kmeans` from package `kmeans`

```
kmeans(x, centers, iter.max = 10, nstart = 1)
```

- *x*: numeric matrix, numeric data frame or a numeric vector
- *centers:* Possible values are the number of clusters (k) or a set of initial cluster centers.

K-means clustering in R

Function `kmeans` from package `kmeans`

```
kmeans(x, centers, iter.max = 10, nstart = 1)
```

- *x*: numeric matrix, numeric data frame or a numeric vector
- *centers:* Possible values are the number of clusters (k) or a set of initial cluster centers.
- *iter.max:* The maximum number of iterations allowed. Default value is 10.

K-means clustering in R

Function `kmeans` from package `kmeans`

```
kmeans(x, centers, iter.max = 10, nstart = 1)
```

- *x*: numeric matrix, numeric data frame or a numeric vector
- *centers:* Possible values are the number of clusters (k) or a set of initial cluster centers.
- *iter.max:* The maximum number of iterations allowed. Default value is 10.
- *nstart:* The number of random starting partitions when centers is a number.

K-means clustering in R

`kmeans()` from package `kmeans` function returns a list including:

- *cluster:* a vector of integers (from `1:k`) indicating the cluster to which each point is allocated.

K-means clustering in R

`kmeans()` from package `kmeans` function returns a list including:

- *cluster:* a vector of integers (from `1:k`) indicating the cluster to which each point is allocated.
- *centers:* a matrix of cluster centers.

K-means clustering in R

`kmeans()` from package `kmeans` function returns a list including:

- *cluster:* a vector of integers (from `1:k`) indicating the cluster to which each point is allocated.
- *centers:* a matrix of cluster centers.
- *withinss:* vector of within-cluster sum of squares, one component per cluster.

K-means clustering in R

`kmeans()` from package `kmeans` function returns a list including:

- *cluster:* a vector of integers (from `1:k`) indicating the cluster to which each point is allocated.
- *centers:* a matrix of cluster centers.
- *withinss:* vector of within-cluster sum of squares, one component per cluster.
- *tot.withinss:* total within-cluster sum of squares. That is, `sum(withinss)`.

K-means clustering in R

`kmeans()` from package `kmeans` function returns a list including:

- *cluster:* a vector of integers (from `1:k`) indicating the cluster to which each point is allocated.
- *centers:* a matrix of cluster centers.
- *withinss:* vector of within-cluster sum of squares, one component per cluster.
- *tot.withinss:* total within-cluster sum of squares. That is, `sum(withinss)`.
- *size:* the number of points in each cluster.

K-means clustering in R

```
set.seed(123)
# Two-dimensional data format
df <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
            matrix(rnorm(100, mean = 1, sd = 0.3),
            ncol = 2))
colnames(df) <- c("x", "y")
head(df,5)
```

```
##                   x            y
## [1,] -0.16814269  0.075995554
## [2,] -0.06905325 -0.008564027
## [3,]  0.46761249 -0.012861137
## [4,]  0.02115252  0.410580685
## [5,]  0.03878632 -0.067731296
```

K-means clustering in R

The R code below performs k-means clustering with k = 2:

```
# Compute k-means
set.seed(123)
km.res <- kmeans(df, 2, nstart = 25)
# Cluster number for each of the observations
km.res$cluster
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

K-means clustering in R

```
# Cluster size
km.res$size
```

```
## [1] 50 50
```

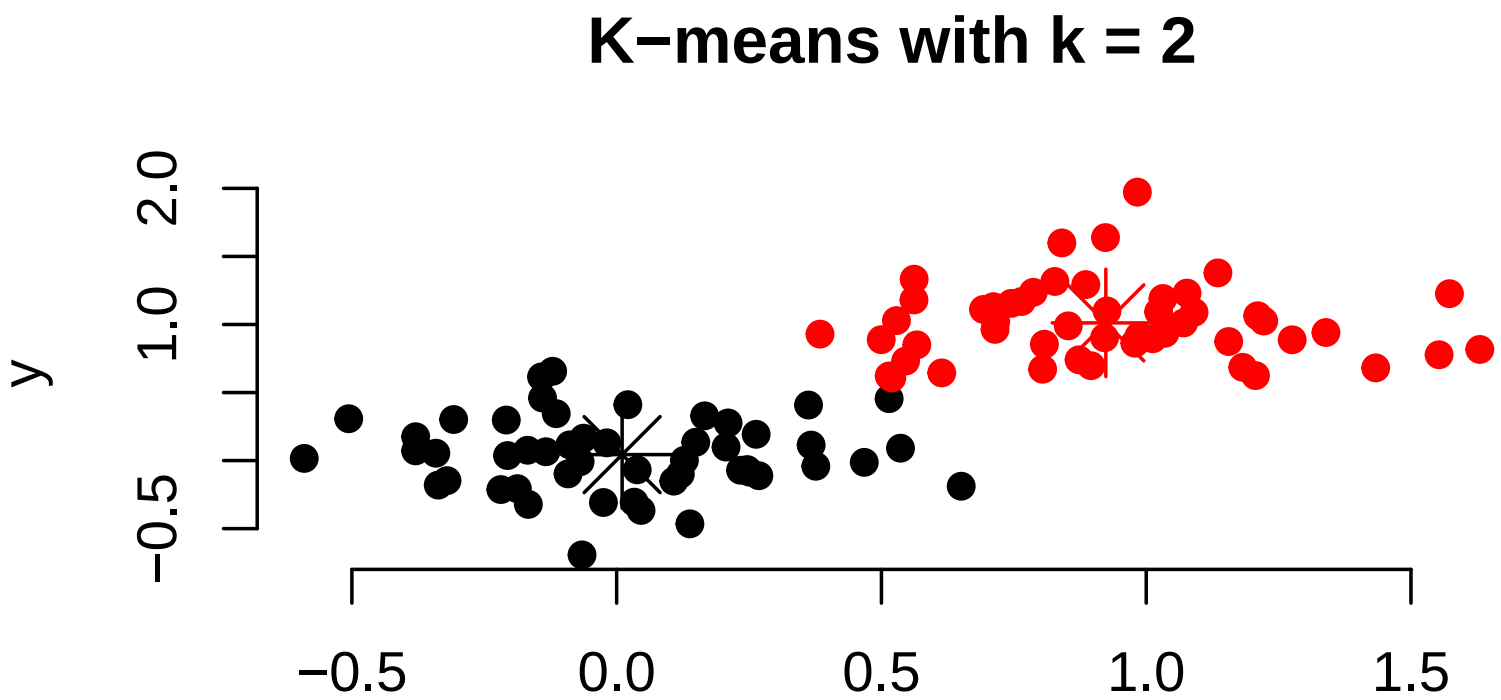
```
# Cluster means
km.res$centers
```

```
##           x          y
## 1 0.01032106 0.04392248
## 2 0.92382987 1.01164205
```

It's possible to plot the data with coloring each data point according to its cluster assignment. The cluster centers are specified using "big stars":

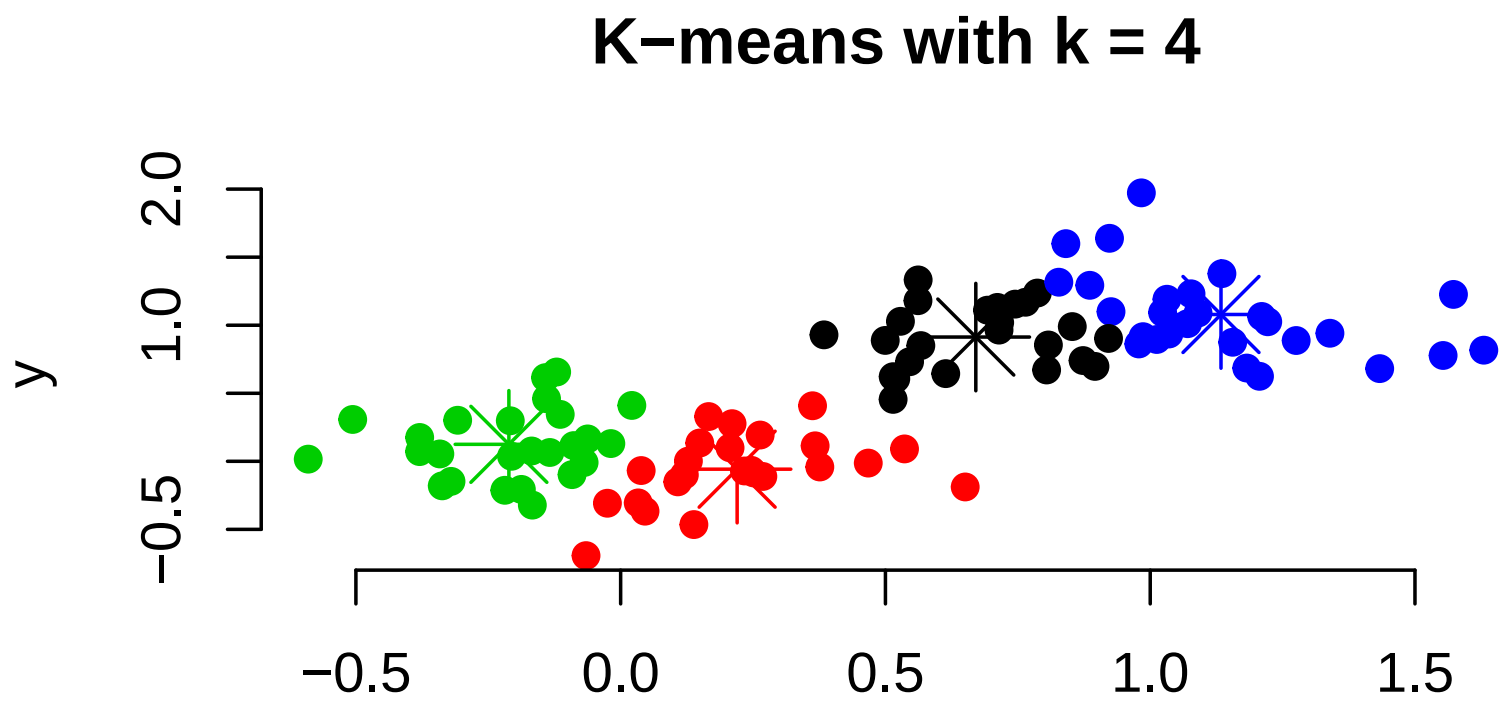
K-means clustering in R

```
plot(df, col = km.res$cluster, pch = 19, frame = FALSE,
     main = "K-means with k = 2")
points(km.res$centers, col = 1:2, pch = 8, cex = 3)
```



K-means clustering in R

```
km.res <- kmeans(df, 4, nstart = 25)
plot(df, col = km.res$cluster, pch = 19, frame = FALSE,
     main = "K-means with k = 4")
points(km.res$centers, col = 1:4, pch = 8, cex = 3)
```



K-means clustering in R

```
# Print the result
km.res
```

```
## K-means clustering with 4 clusters of sizes 24, 24, 25,
##
## Cluster means:
##             x           y
## 1  0.6706931  0.91293798
## 2  0.2199345 -0.05766457
## 3 -0.2110757  0.12500530
## 4  1.1336807  1.07876045
##
## Clustering vector:
##   [1] 3 3 2 3 2 1 2 3 3 3 2 2 2 2 3 2 2 3 2 3 3 2 3 3 3
##  [36] 2 2 3 3 3 3 3 3 2 2 3 3 3 2 2 1 4 4 1 1 4 1 1 4 4
##  [71] 4 1 1 4 4 1 4 4 1 4 4 4 4 1 1 4 1 4 4 1 4 1 1 1 1
```

K-means clustering in R

```
set.seed(123)
# K-means with nstart = 1
km.res <- kmeans(df, 4, nstart = 1)
km.res$tot.withinss
```

```
## [1] 10.13198
```

```
# K-means with nstart = 25
km.res <- kmeans(df, 4, nstart = 25)
km.res$tot.withinss
```

```
## [1] 9.814517
```

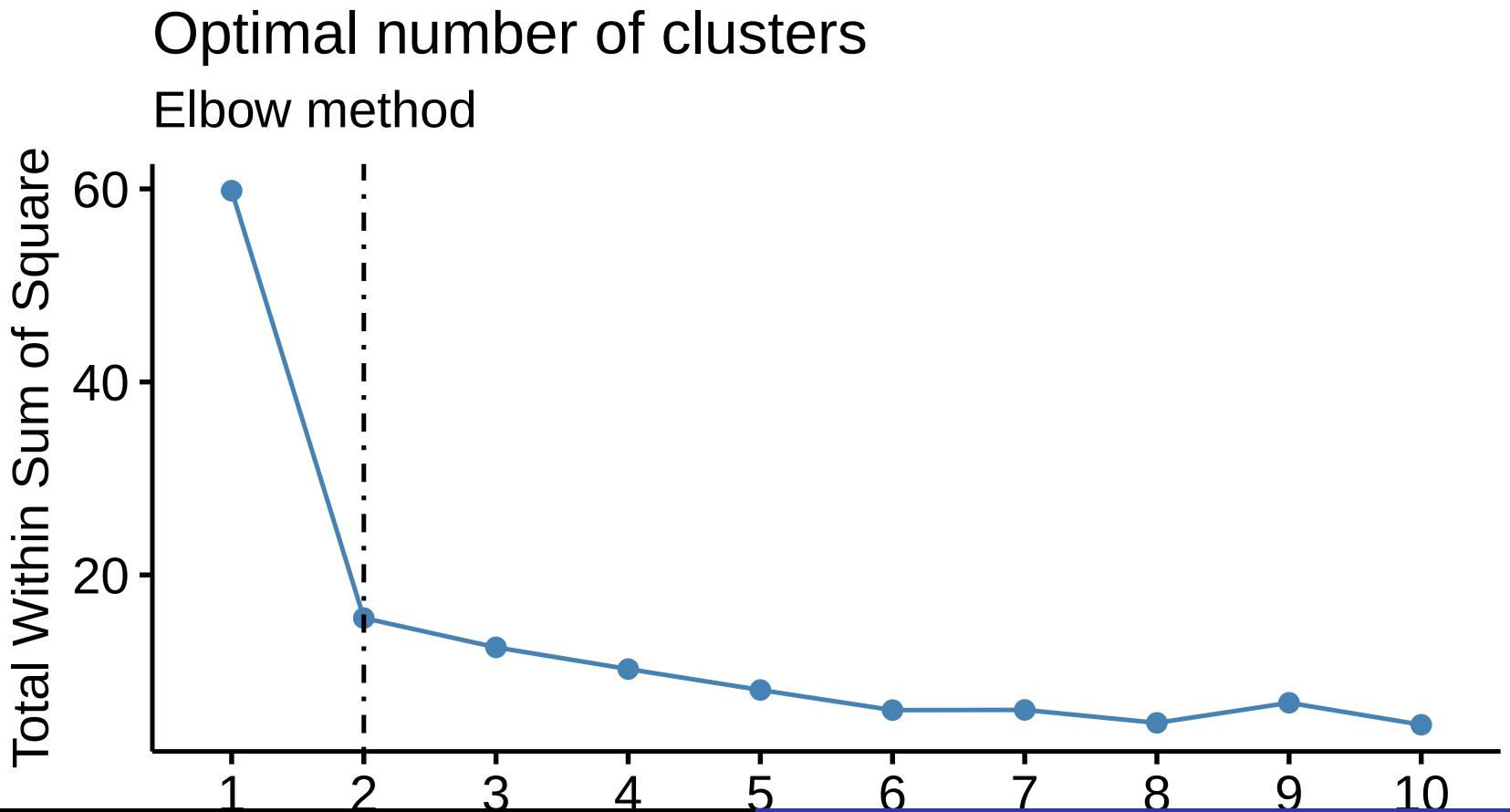
Predict memberships

```r
#load the libeary clue and use the function cl_predict
library(clue)
nr  <- nrow(df)
#choose a random 90% sample
ind <- sample(nr, 0.9 * nr, replace = FALSE)
km.res <- kmeans(df[ind, ], 2, nstart = 25)
cl_predict(km.res, df[-ind, ])
```

```
## Class ids:
##  [1] 1 1 1 1 1 1 2 2 2 2
```

How to choose k

```
fviz_nbclust(df, kmeans, method = "wss") +
  geom_vline(xintercept = 2, linetype = 4) +
  labs(subtitle = "Elbow method")
```



Drawbacks of K-means

1. Need to choose the right `k`

Pleae complete clustering_algorithms_lab.Rmd (kmeans section)

Drawbacks of K-means

1. Need to choose the right `k`
2. Cannot Handle Noise Data and Outliers

Pleae complete clustering_algorithms_lab.Rmd (kmeans section)

Drawbacks of K-means

1. Need to choose the right `k`
2. Cannot Handle Noise Data and Outliers
3. Cannot Handle Non-spherical Data

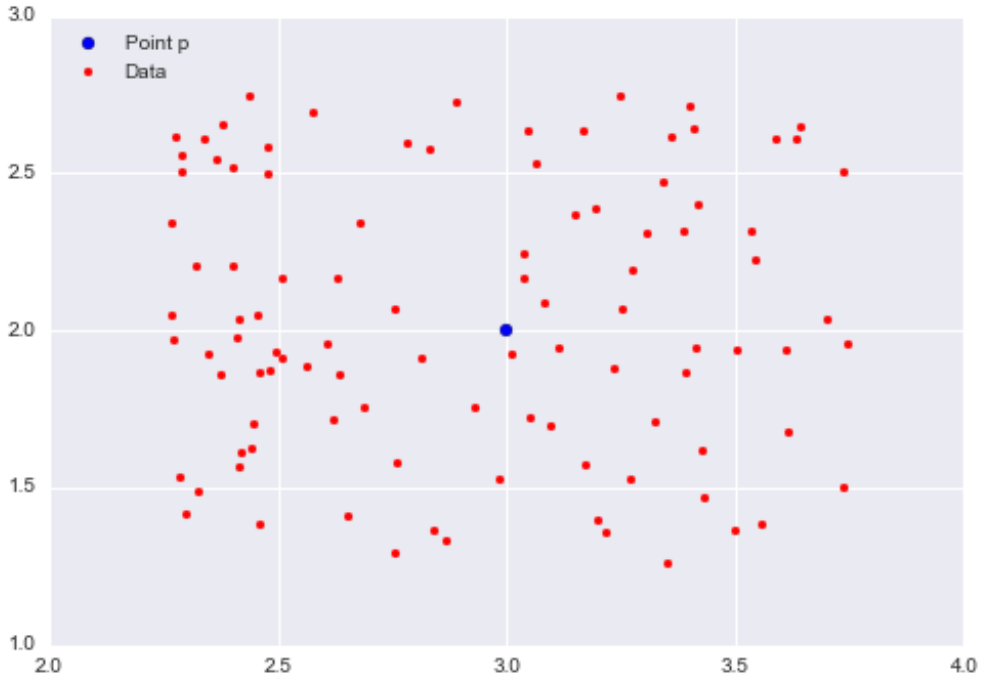
Pleae complete clustering_algorithms_lab.Rmd (kmeans section)

DBSCAN: Density-Based Spatial Clustering Algorithm with Noise

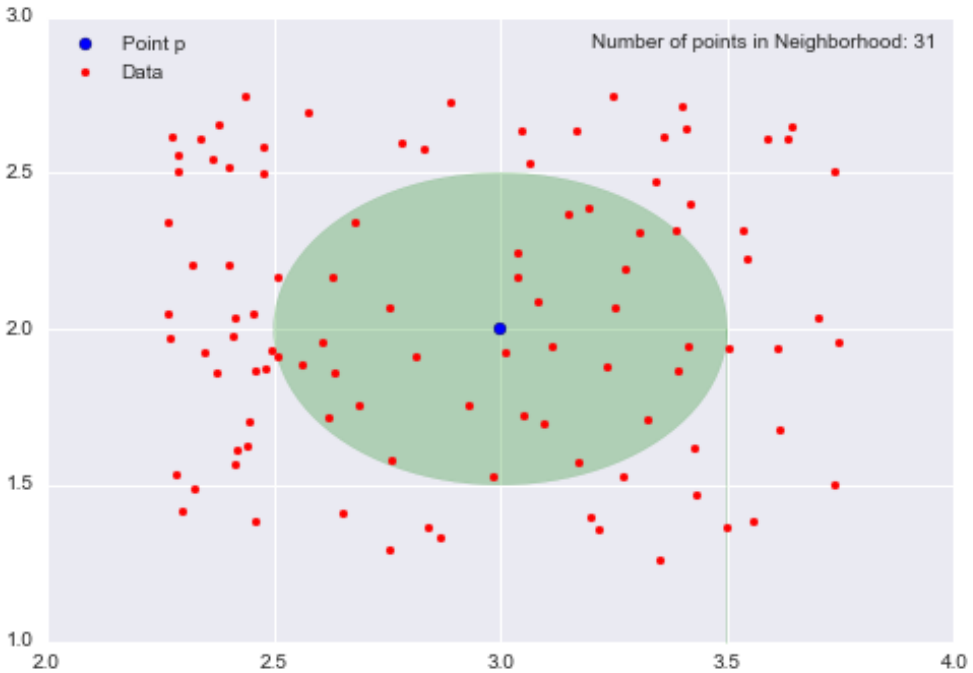
Preliminary: ϵ-Balls and neighborhood density

For some $\epsilon > 0$ and some point p, the ϵ-neighborhood of p is defined as the set of points that are at most distance ϵ away from p.

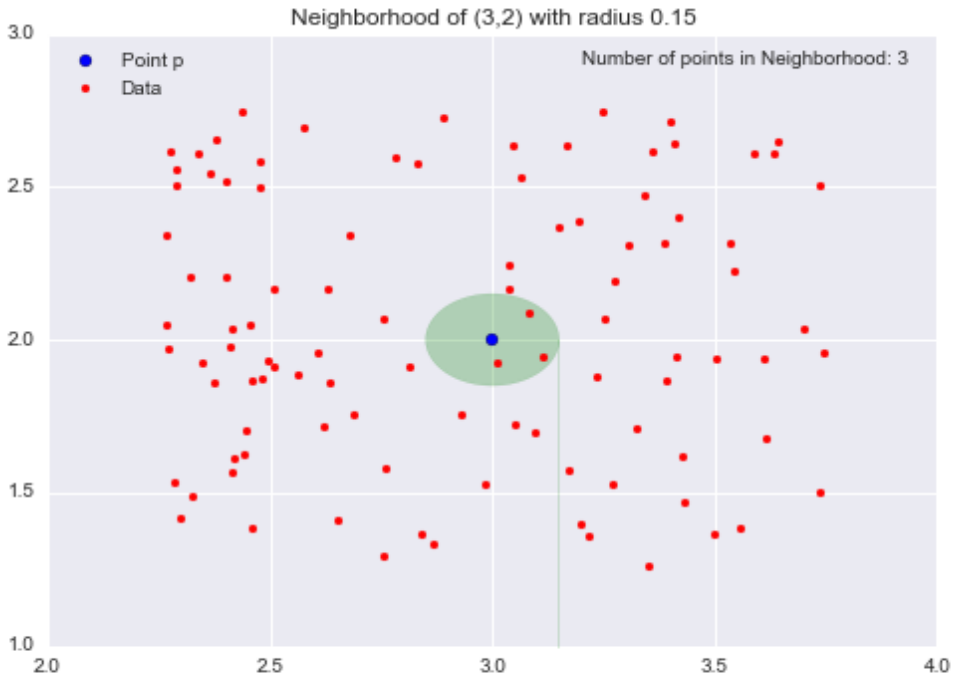
Preliminary: ϵ-Balls and neighborhood density



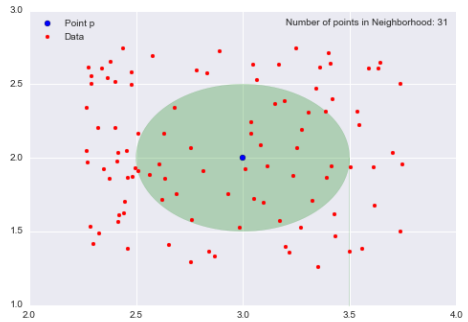
Preliminary: ϵ-Balls and neighborhood density



Preliminary: ϵ-Balls and neighborhood density



Preliminary: ϵ-Balls and neighborhood density



$$density = \frac{mass}{volume}$$

$mass = 31.$

$volume = \pi \times 0.5^2 = \frac{\pi}{4}$

Therefore, our local density approximation at $p = (3, 2)$ is calculated as $density = mass/volume = 31/\frac{\pi}{4} = 124/\pi \ = 39.5.$

Preliminary: ϵ-Balls and neighborhood density

- This value is meaningless by itself, but if we calculate the local density approximation for all points in our dataset

While this is not exactly what DBSCAN does, it forms the general intuition behind density-based clustering.

Preliminary: ϵ-Balls and neighborhood density

- This value is meaningless by itself, but if we calculate the local density approximation for all points in our dataset
- we could cluster our points by saying that points that are nearby (contained in the same neighborhood) and have similar local density approximations belong in the same cluster.If we decrease the value of ϵ we can construct smaller neighborhoods (less volume) that would also contain fewer data points.

While this is not exactly what DBSCAN does, it forms the general intuition behind density-based clustering.

Preliminary: ϵ-Balls and neighborhood density

- This value is meaningless by itself, but if we calculate the local density approximation for all points in our dataset
- we could cluster our points by saying that points that are nearby (contained in the same neighborhood) and have similar local density approximations belong in the same cluster.If we decrease the value of ϵ we can construct smaller neighborhoods (less volume) that would also contain fewer data points.
- Ideally, we want to identify highly dense neighborhoods where most of the data points are contained in these neighborhoods, but the volume of each of these neighborhoods is relatively small.

While this is not exactly what DBSCAN does, it forms the general intuition behind density-based clustering.

DBSCAN

The ϵ-neighborhood is fundamental to DBSCAN to approximate local density, so the algorithm has two parameters:

- ϵ: The radius of our neighborhoods around a data point p. minPts: The minimum number of data points we want in a neighborhood to define a cluster.

Using these two parameters, DBSCAN categories the data points into three categories:

DBSCAN

The ϵ-neighborhood is fundamental to DBSCAN to approximate local density, so the algorithm has two parameters:

- ϵ: The radius of our neighborhoods around a data point p. minPts: The minimum number of data points we want in a neighborhood to define a cluster.
- *minPts*: The minimum number of data points we want in a neighborhood to define a cluster.

Using these two parameters, DBSCAN categories the data points into three categories:

DBSCAN

- *Core Points*: A data point p is a core point if **Nbhd**(p, ϵ) [$\epsilon-$neighborhood of p] contains at least minPts ; **Nbhd**(p, ϵ) $>=$ *minPts*.

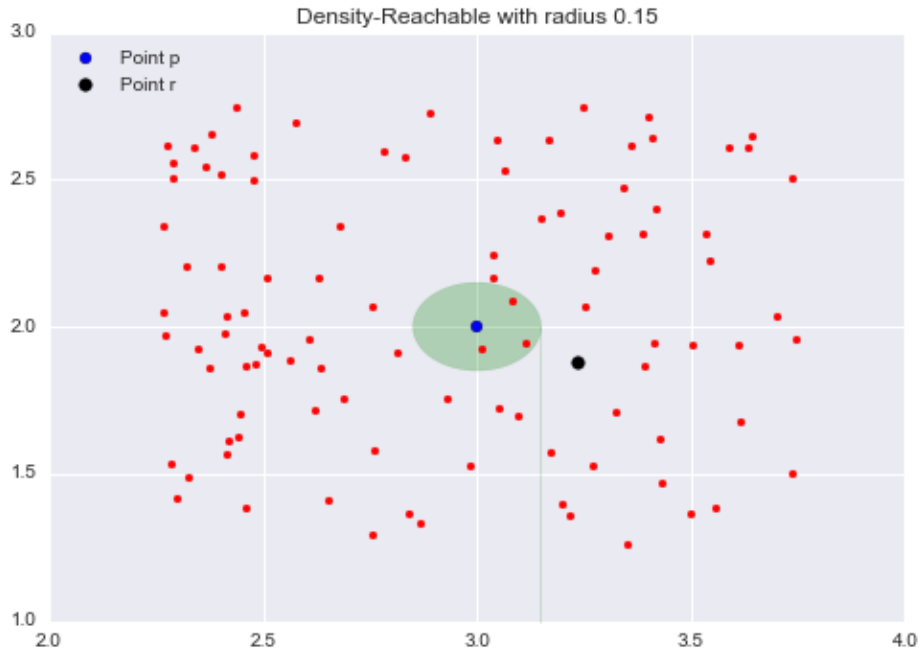
DBSCAN

- *Core Points*: A data point p is a core point if **Nbhd**(p, ϵ) [$\epsilon-$neighborhood of p] contains at least minPts ; **Nbhd**(p, ϵ) $>=$ *minPts*.
- *Border Points*: A data point q is a border point if *Nbhd*(p, ϵ) contains less than *minPts* data points, but q is reachable from some core point p.

DBSCAN

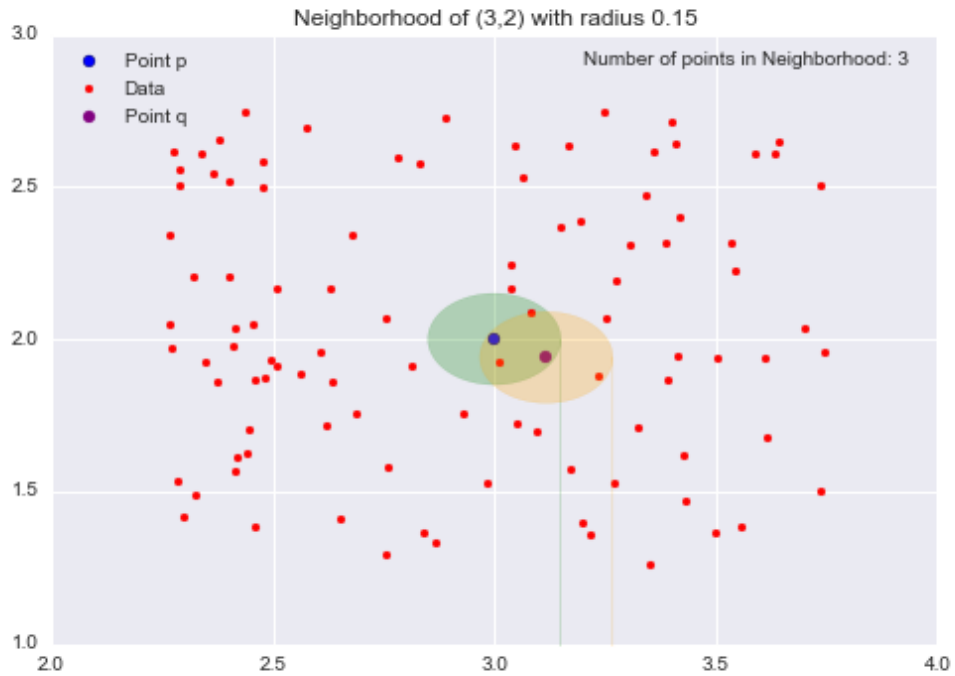
- *Core Points*: A data point p is a core point if **Nbhd**(p, ϵ) [$\epsilon-$neighborhood of p] contains at least minPts ; **Nbhd**(p, ϵ) $>=$ *minPts*.
- *Border Points*: A data point q is a border point if *Nbhd*(p, ϵ) contains less than *minPts* data points, but q is reachable from some core point p.
- *Outlier*: A data point O is an outlier if it is neither a *core point* nor a *border point*. Essentially, this is the "other" class.

DBSCAN: Border Points

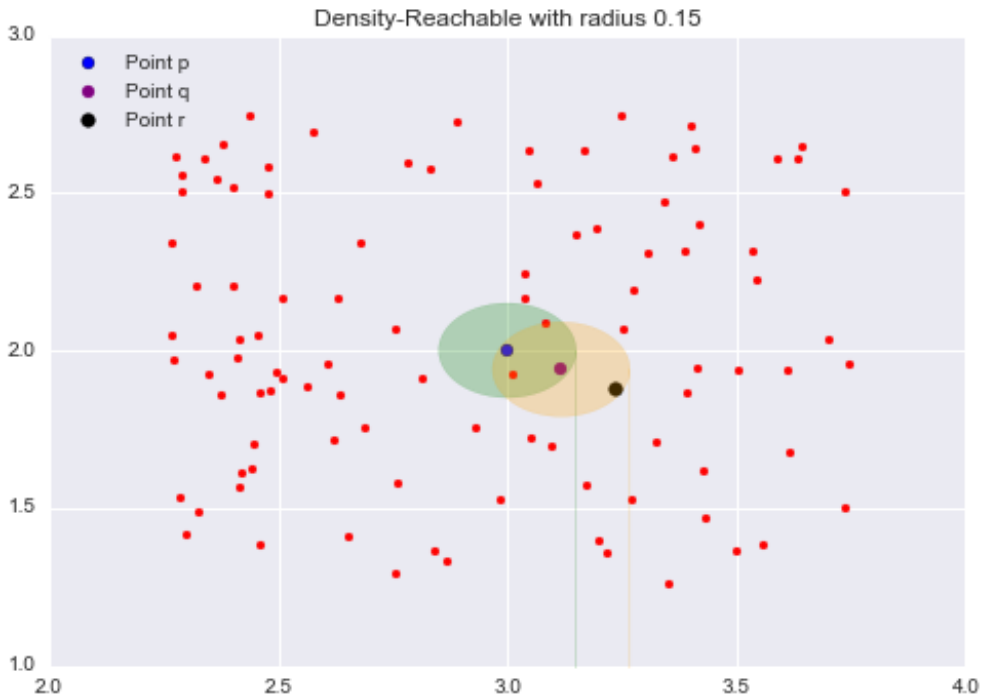


The three points in **Nbhd**(p, ϵ) are said to be directly reachable from p

DBSCAN: Border Points



DBSCAN: Border Points



If I 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: Outliers

Outliers are points that are neither core points nor are they close enough to a cluster to be *density-reachable* from a core point. Outliers are not assigned to any cluster and, depending on the context, may be considered anomalous points.

Now that I have covered all the preliminaries, we can finally talk about how the algorithm works in practice.

DBSCAN algorithm

- Pick a point at random that has not been assigned to a cluster or been designated as an outlier. Compute its neighborhood to determine if it's a core point. If yes, start a cluster around this point. If no, label the point as an outlier.

DBSCAN algorithm

- Pick a point at random that has not been assigned to a cluster or been designated as an outlier. Compute its neighborhood to determine if it's a core point. If yes, start a cluster around this point. If no, label the point as an outlier.
- Once we find a core point and thus a cluster, expand the cluster by adding all directly-reachable points to the cluster. Perform "neighborhood jumps" to find all density-reachable points and add them to the cluster. If an an outlier is added, change that point's status from outlier to border point.

DBSCAN algorithm

- Pick a point at random that has not been assigned to a cluster or been designated as an outlier. Compute its neighborhood to determine if it's a core point. If yes, start a cluster around this point. If no, label the point as an outlier.
- Once we find a core point and thus a cluster, expand the cluster by adding all directly-reachable points to the cluster. Perform "neighborhood jumps" to find all density-reachable points and add them to the cluster. If an an outlier is added, change that point's status from outlier to border point.
- Repeat these two steps until all points are either assigned to a cluster or designated as an outlier.

DBSCAN R implementation

We need the packages `fpc;dbscan` and `factoextra`

To determine the optimal value of ϵ we use the functionwe use

```
dbscan::kNNdistplot(data, k = minPts)
```

Compute DBSCAN using `fpc::dbscan()` or `dbscan::dbscan()`.

```
# fpc package
res.fpc <- fpc::dbscan(data, eps = espsilon, MinPts = k)
# dbscan package
res.db <- dbscan::dbscan(data, eps = espsilon, MinPts = k)
```

DBSCAN R implementation

- The result of the function `fpc::dbscan()` and provides an object of class 'dbscan' containing the following components:

DBSCAN R implementation

- The result of the function `fpc::dbscan()` and provides an object of class 'dbscan' containing the following components:
  - cluster: integer vector coding cluster membership with noise observations (singletons) coded as 0

DBSCAN R implementation

- The result of the function `fpc::dbscan()` and provides an object of class 'dbscan' containing the following components:
  - cluster: integer vector coding cluster membership with noise observations (singletons) coded as 0
  - isseed: logical vector indicating whether a point is a seed (not border, not noise)

DBSCAN R implementation

- The result of the function `fpc::dbscan()` and provides an object of class 'dbscan' containing the following components:
  - cluster: integer vector coding cluster membership with noise observations (singletons) coded as 0
  - isseed: logical vector indicating whether a point is a seed (not border, not noise)
  - eps: parameter eps

DBSCAN R implementation

- The result of the function `fpc::dbscan()` and provides an object of class 'dbscan' containing the following components:
  - cluster: integer vector coding cluster membership with noise observations (singletons) coded as 0
  - isseed: logical vector indicating whether a point is a seed (not border, not noise)
  - eps: parameter eps
  - MinPts: parameter MinPts

DBSCAN R implementation

- The result of the function `fpc::dbscan()` and provides an object of class 'dbscan' containing the following components:
  - cluster: integer vector coding cluster membership with noise observations (singletons) coded as 0
  - isseed: logical vector indicating whether a point is a seed (not border, not noise)
  - eps: parameter eps
  - MinPts: parameter MinPts
- The result of the function dbscan::dbscan() is an integer vector with cluster assignments. Zero indicates noise points.

DBSCAN R implementation example

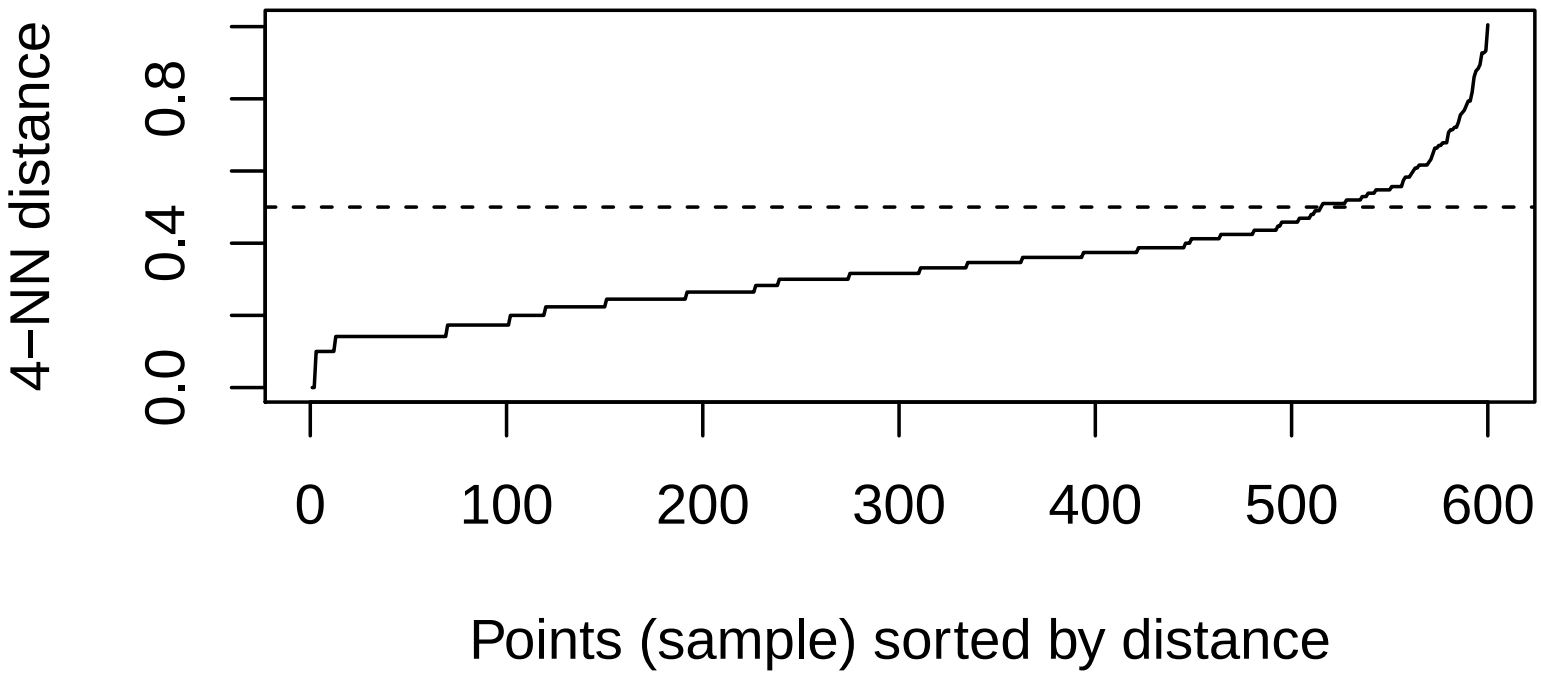
```
# Load the data
data("iris")
iris <- as.matrix(iris[, 1:4])

# The optimal value of "eps" parameter can be determined as

dbscan::kNNdistplot(iris, k =  4)
abline(h = 0.5, lty = 2)
# Compute DBSCAN using fpc::dbscan() and dbscan::dbscan().

set.seed(123)
# fpc package
res.fpc <- fpc::dbscan(iris, eps = 0.4, MinPts = 4)
# dbscan package
res.db <- dbscan::dbscan(iris, 0.4, 4)
```

DBSCAN R implementation example



Pleae complete clustering_algorithms_lab.Rmd (DBSCAN section)