Distances and Clustering

CSAMA 2026

Davide Risso

Table of contents

Introduction

Unsupervised classification

The goal of clustering is to classify observations into discrete groups, without knowing the group labels, *nor the number of groups*.

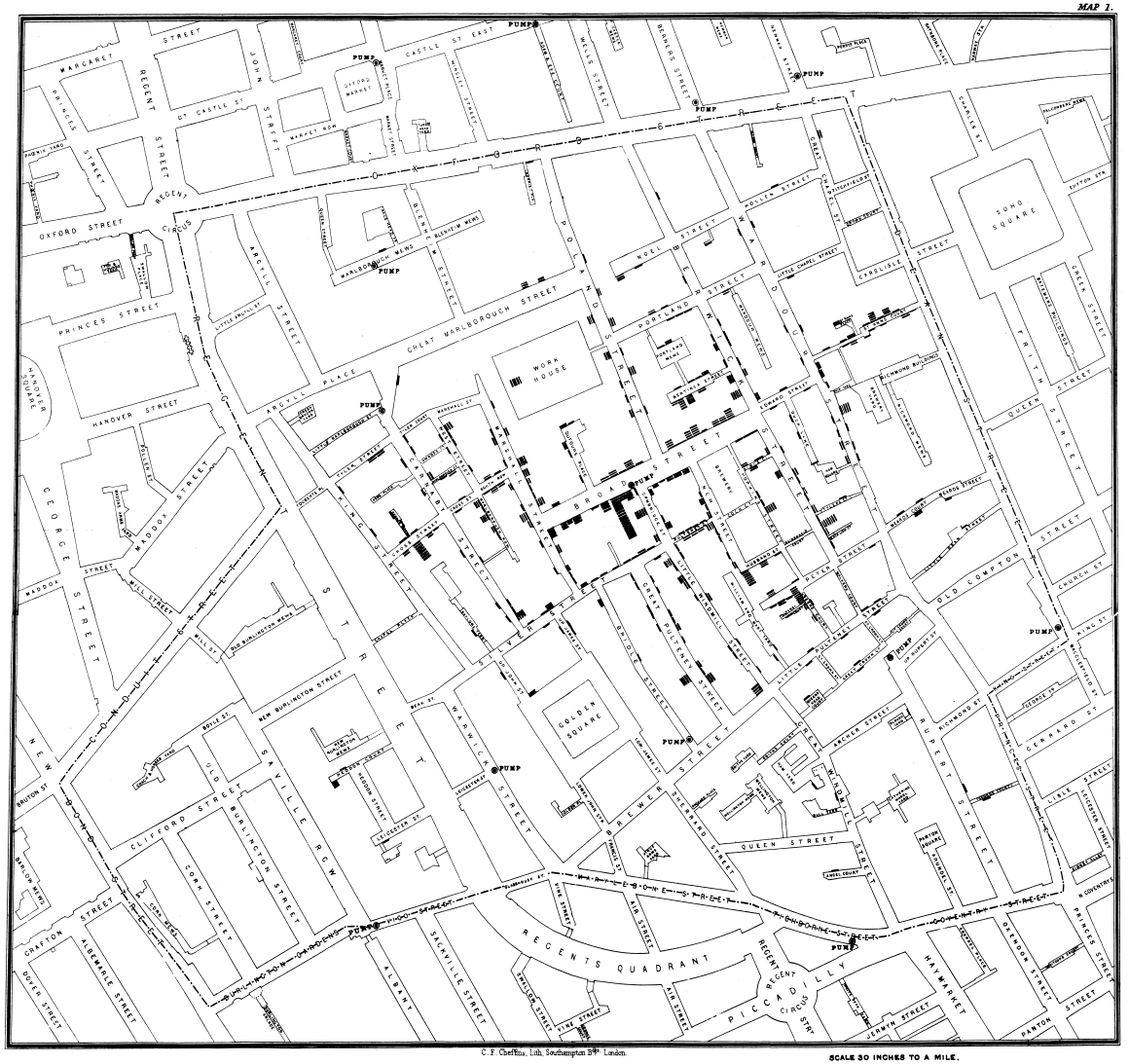
Why clustering?

Clustering problems are often posed in a vague way: *“let’s group the data and see what the groups tell us.”*

Other times, this is done to simplify complex measurements: for instance, by grouping tumor samples we can define *molecular subtypes* that can be used for treatment decision making.

In yet other instances, we conceptualize our observations as noisy versions of idealized objects: e.g., we hope that by clustering single cells, our groups represent cell types.

Clustering as a discovery tool



John Snow’s cholera map

Clustering as a discovery tool

In 1855, the English physiscian John Snow made a map of cholera cases and identified spatial *clusters* of cases.

The proximity of dense clusters of cases to the Broadstreet pump pointed to its water as a possible culprit.

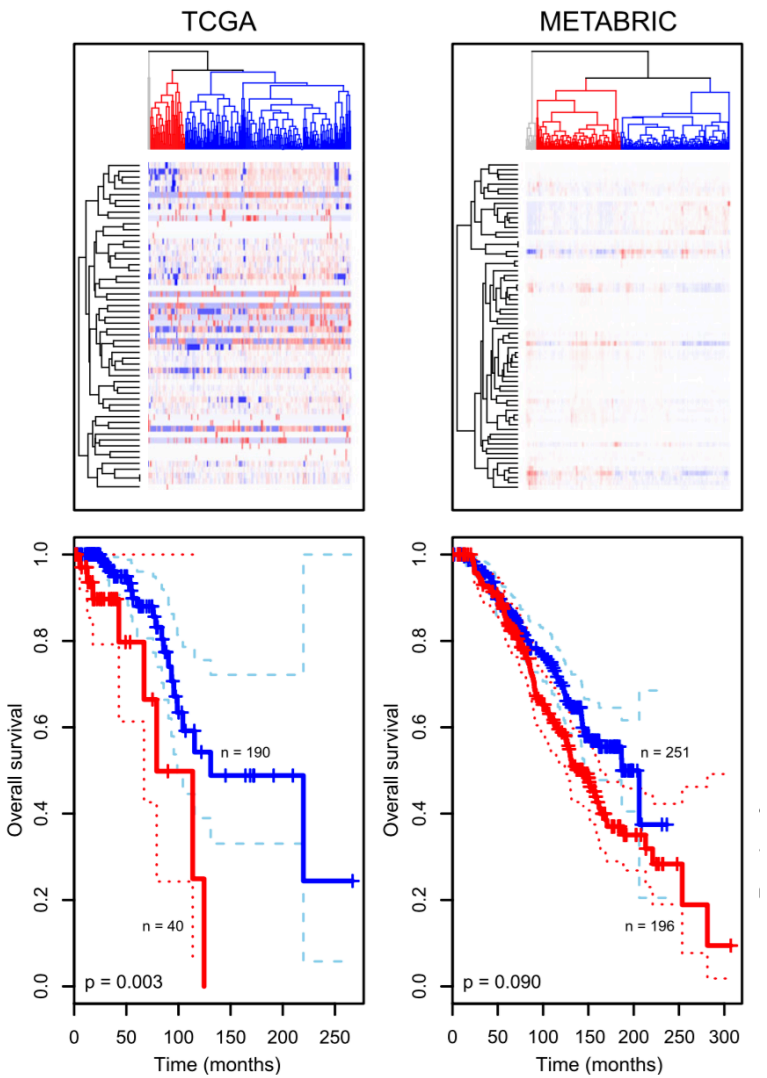
Clustering enabled him to infer the source of the cholera outbreak.

Clustering as a discovery tool

Another example is how clustering has enabled researchers to enhance their understanding of cancer biology.

Tumors that appear to be the same based on their anatomical location and histopathology can be grouped into multiple clusters based on their molecular signatures, such as gene expression data.

Clustering as a discovery tool



Aure et al. (2017)

Intuition of clustering

Intuitively, with clustering we try to group together **similar observations**.

But how do we define similarity?

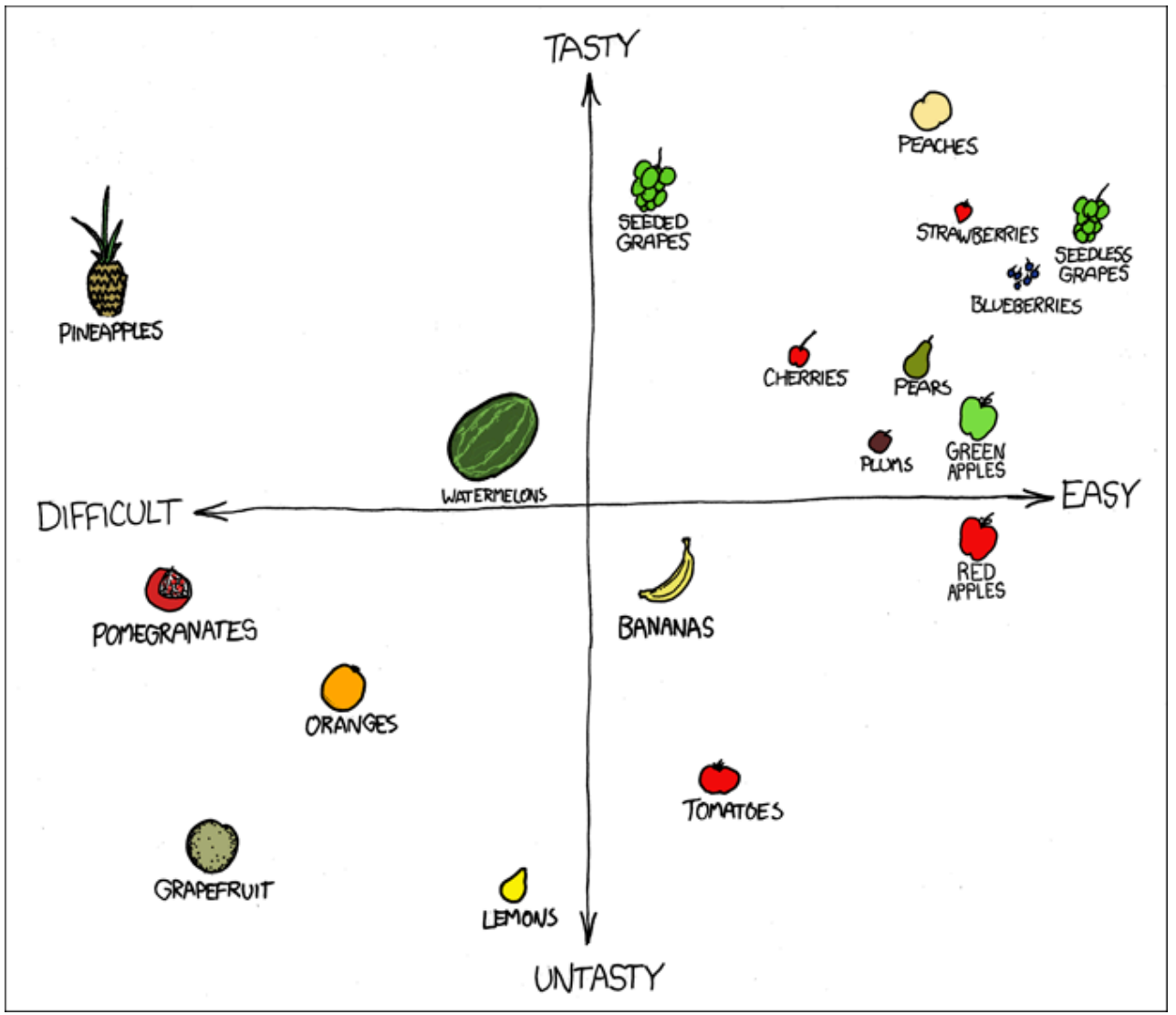
Similarity

Credit: Holmes and Huber MSMB

Are the rows or the columns more similar to each other?

Distance

Distance as a measure of similarity



https://xkcd.com/388/

Distance as dissimilarity

We can think of observations as points in a multi-dimensional space defined by the variables that we measure on them.

It is intuitive to think that by measuring the distance between object in such a space will provide a way to group them.

But… **this is easier said than done!**

Where do we start…?

- Do we need all the variables or only a subset of them?
- Do we need to transform the variables?
  - Are they on the same scale?
  - Do they have a very skewed distribution?
  - Should they all contribute equally to the clustering? (cf. standardization)
- Which distance should I use?

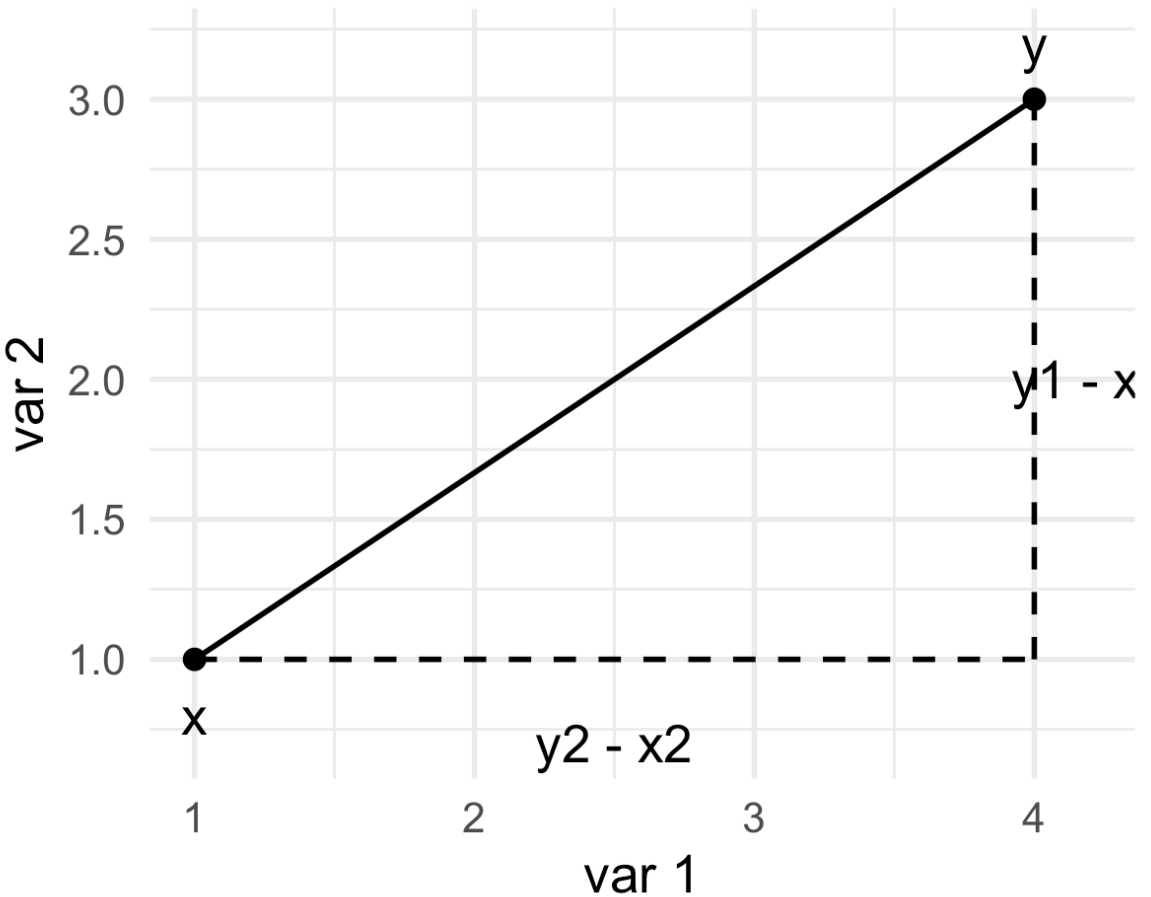
Euclidean distance

Definition

The Euclidean distance between two points $x = [x_1 \ \ldots \ x_p]$ and $y = [y_1 \ \ldots \ y_p]$ in a p-dimensional space is the square root of the sum of squares of the differences in all coordinate directions:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_p - y_p)^2}.$$

Euclidean distance



$$d(x, y)^2 = (y_1 - x_1)^2 + (y_2 - x_2)^2.$$

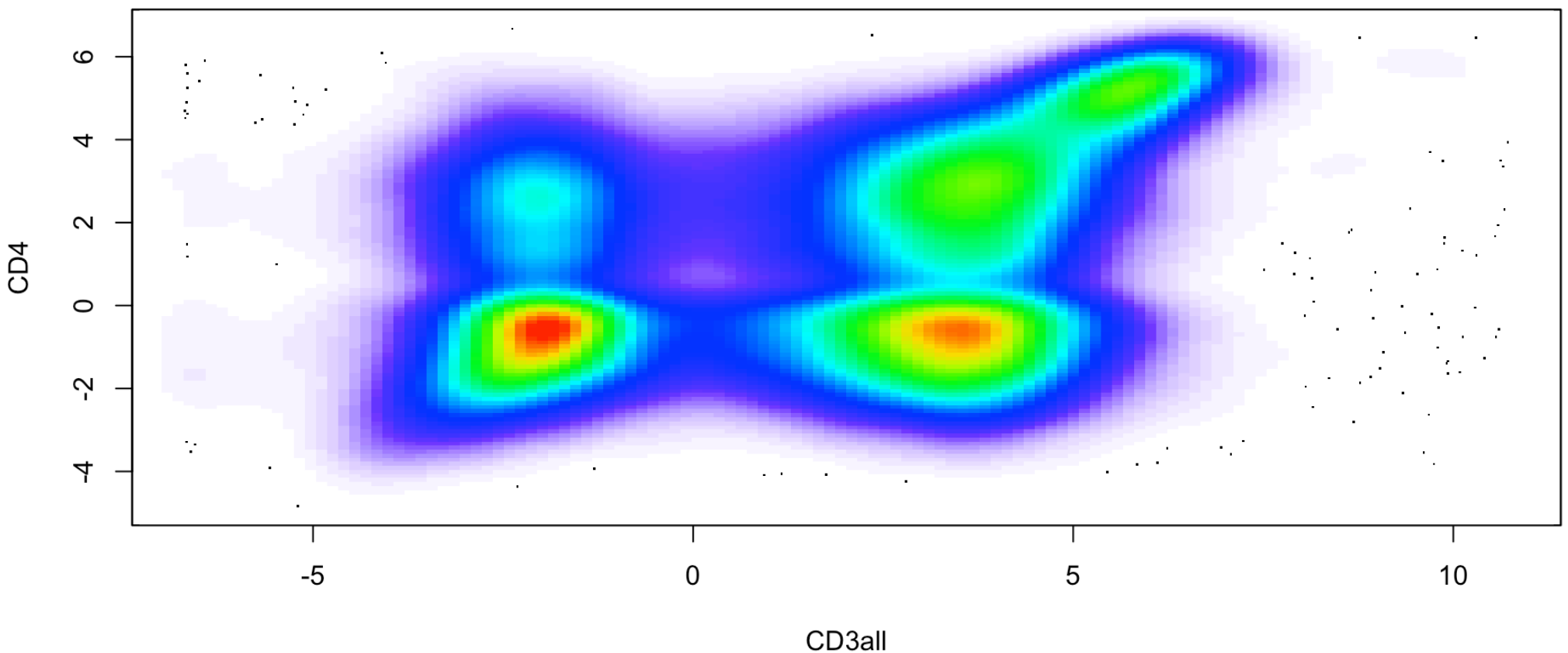
Example: flow cytometry

At different stages of their development, immune cells express unique combinations of proteins on their surfaces. These protein-markers can be collected by flow cytometry (using fluorescence).

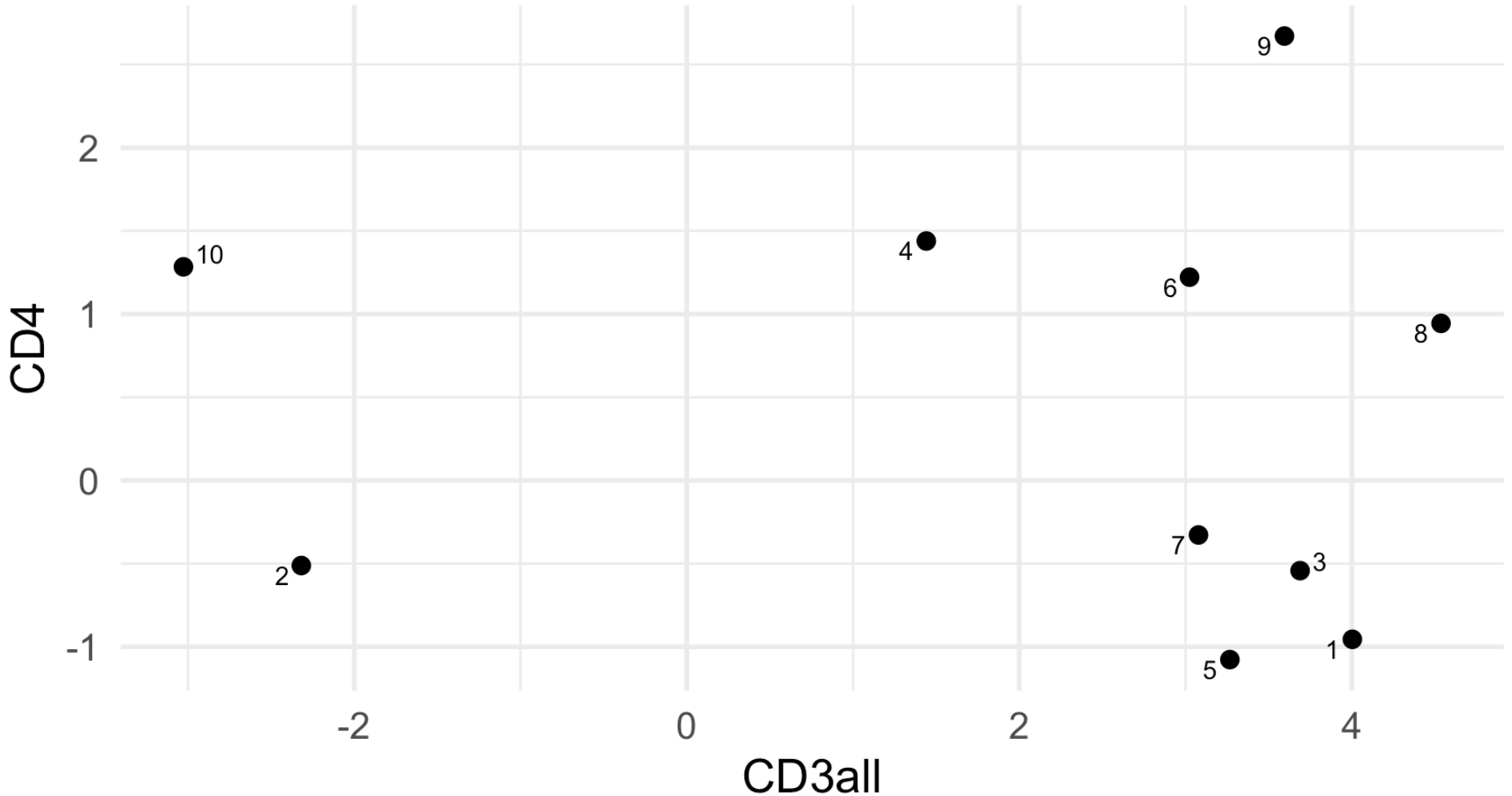
An example of a commonly used marker is CD4, which is expressed by helper T cells. Another one is CD3, which is expressed by all mature T cells. Hence, we can define CD4+ T cells as those cells that express both CD3 and CD4 markers.

Our dataset includes 91,392 cells and 41 markers.

Example: flow cytometry



Flow cytometry (first 10 cells)



Flow cytometry (first 10 cells)

Compute the distance in R.

```
1 dist(cd10)
```

```
          1         2         3         4         5         6         7         8         9
2  6.3364670
3  0.5185929 6.0068202
4  3.5061729 4.2343690 2.9962003
5  0.7465485 5.6129138 0.6811223 3.1079040
6  2.3861852 5.6159429 1.8846760 1.5981157 2.3106538
7  1.1177214 5.3987623 0.6476282 2.4082368 0.7722078 1.5500454
8  1.9719727 7.0071081 1.7106420 3.1350039 2.3862557 1.5376895 1.9350668
9  3.6478490 6.7142556 3.2137253 2.4811919 3.7610604 1.5571473 3.0421742 1.9666198
10 7.3774606 1.9297734 6.9592771 4.4701481 6.7211594 6.0510622 6.3134992 7.5707866 6.7650431
```

Other distances

The Euclidean distance is a natural choice, but not the only options.

There are many different distance metrics that we can use, here we see a few options.

Hamming distance

Not all data are numbers!

For instance, we may be interested in clustering DNA sequences (e.g., in evolutionary biology).

The simplest way to compute distances between character strings is the *Hamming distance* (aka the Edit distance): it simply counts the number of differences between two character strings.

Jaccard distance

When we have binary data, e.g., presence / absence of a certain feature, co-occurence is often more informative than co-absence, espetially when the events are rare.

Let's call f_{11} the number of times a feature co-occurs both in x and y, f_{10} (and f_{01}) the number of times a feature occurs in x but not in y (and vice versa), and f_{00} the number of times a feature is co-absent.

The Jaccard Index is

$$J(x, y) = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}.$$

Jaccard Index

Note that the Jaccard Index corresponds to the size of the *intersection* of two sets divided by the size of their *union* and is a natural way of measuring the similarity of two sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

The Jaccard distance is

$$d(x,y) = 1 - J(x,y).$$

Partitional methods

k-means

K-means is probably the most widely used clustering algorithm.

It starts from pre-specifying the number of clusters, k.

The main idea is to *partition* the n observations into k groups so that the groups are those that **minimize the within-group sum of squares** (i.e., the within-group Euclidean distance).

k-means

Naively, one could try *all possible groups* and choose the one that minimizes the sum of squares.

However, we have $\binom{n}{k}$ possible ways to group n observations into k groups.

> **Note**
>
> Note that $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

```
choose(10, 4)
```

```
[1] 210
```

```
choose(100, 4)
```

```
[1] 3921225
```

```
choose(1000, 4)
```

```
[1] 41417124750
```

k-means algorithm

1. Randomly initialize K centroids, $\mu_1, \ldots, \mu_K$.
2. Assign each observation i to the centroid k that minimizes

$$c_i = \arg\min_k \sum_{j=1}^{p} (x_{ij} - \mu_{kj})^2 = ||x_i - \mu_k||^2.$$

3. For each group k, compute the new centroid as

$$\mu_k = \frac{\sum_{i=1}^{n} \mathbb{1}_{c_i=k} x_i}{\sum_{i=1}^{n} \mathbb{1}_{c_i=k}},$$

where

$$\mathbb{1}_{c_i=k} = \begin{cases} 1 & \text{if } c_i = k, \\ 0 & \text{otherwise.} \end{cases}$$

Example: k-means

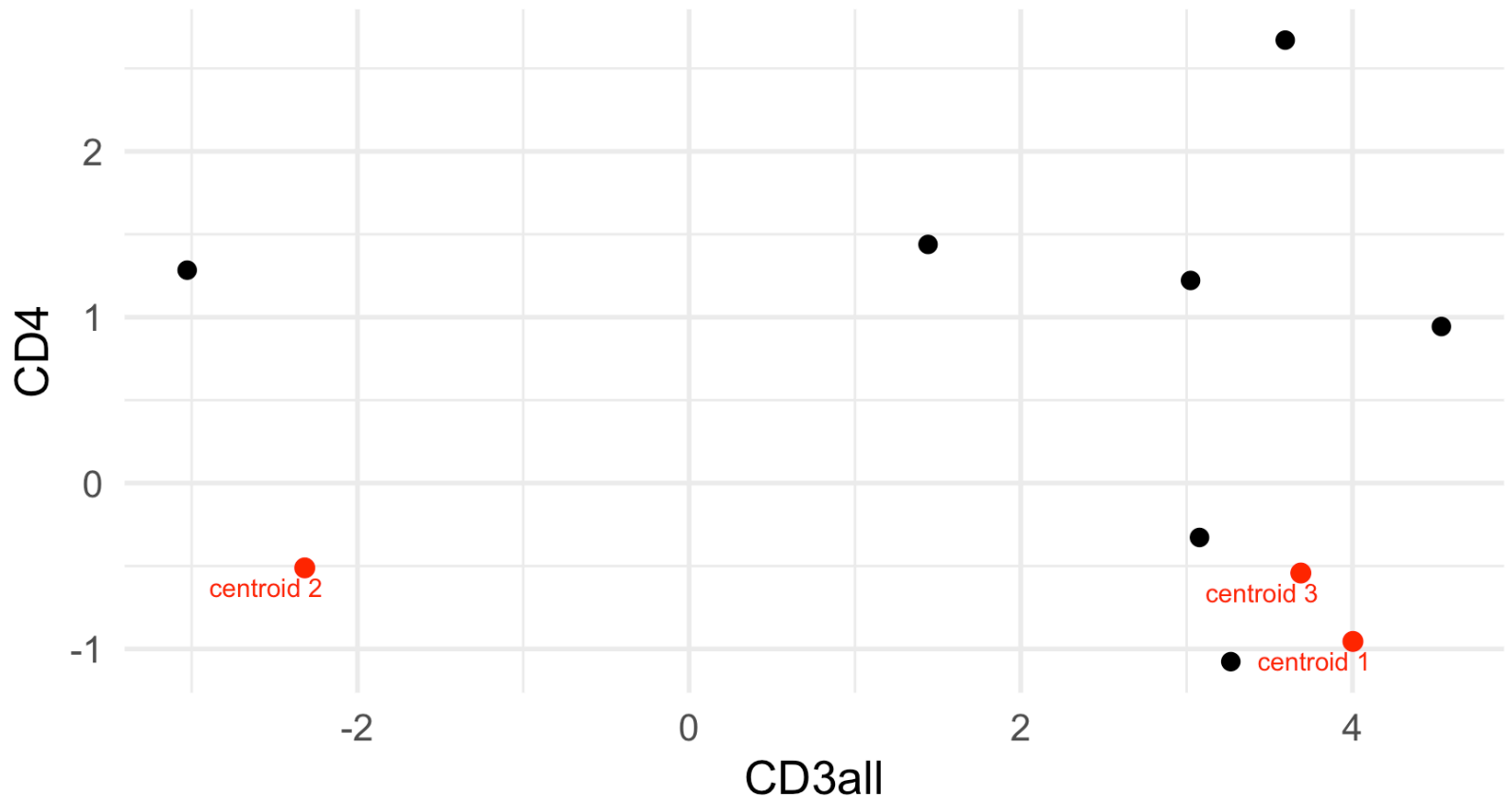
```r
assign_centroids <- function(x, centroids) {
  apply(x, 1, function(xi) {
    which.min(apply(centroids, 1, function(mu) sum((xi - mu)^2)))
  })
}

compute_centroids <- function(x, labels, k) {
  t(vapply(1:k, function(i) colMeans(x[labels == i,]), numeric(ncol(x))))
}
```

Example: k-means

```
1 # pick random centroids
2 (centroids <- cd10[1:3,])
```

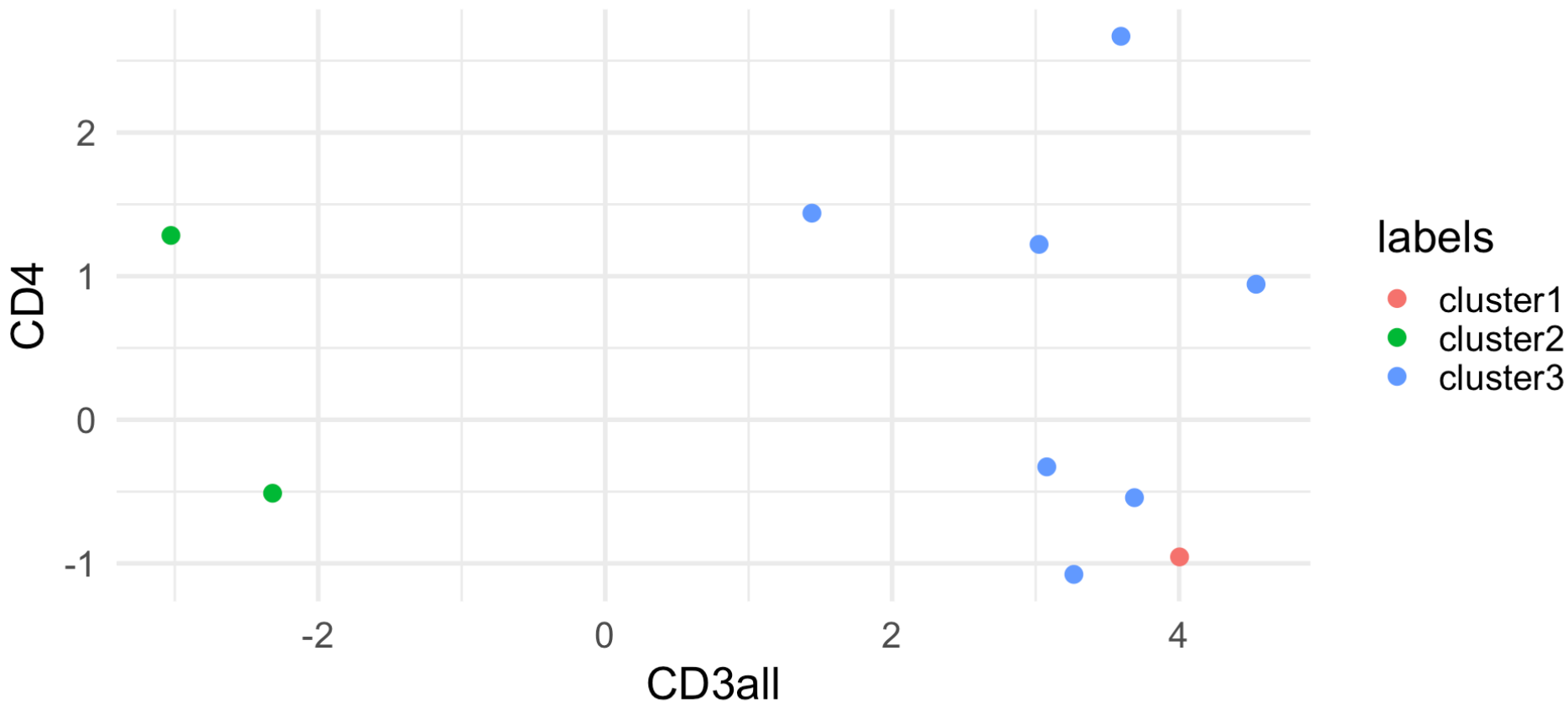
```
     CD3all        CD4
1  4.002830 -0.9547135
2 -2.318107 -0.5113526
3  3.688635 -0.5421367
```



Example: k-means

```
1  # assign observations to centroids
2  (labels <- assign_centroids(cd10, centroids))
```

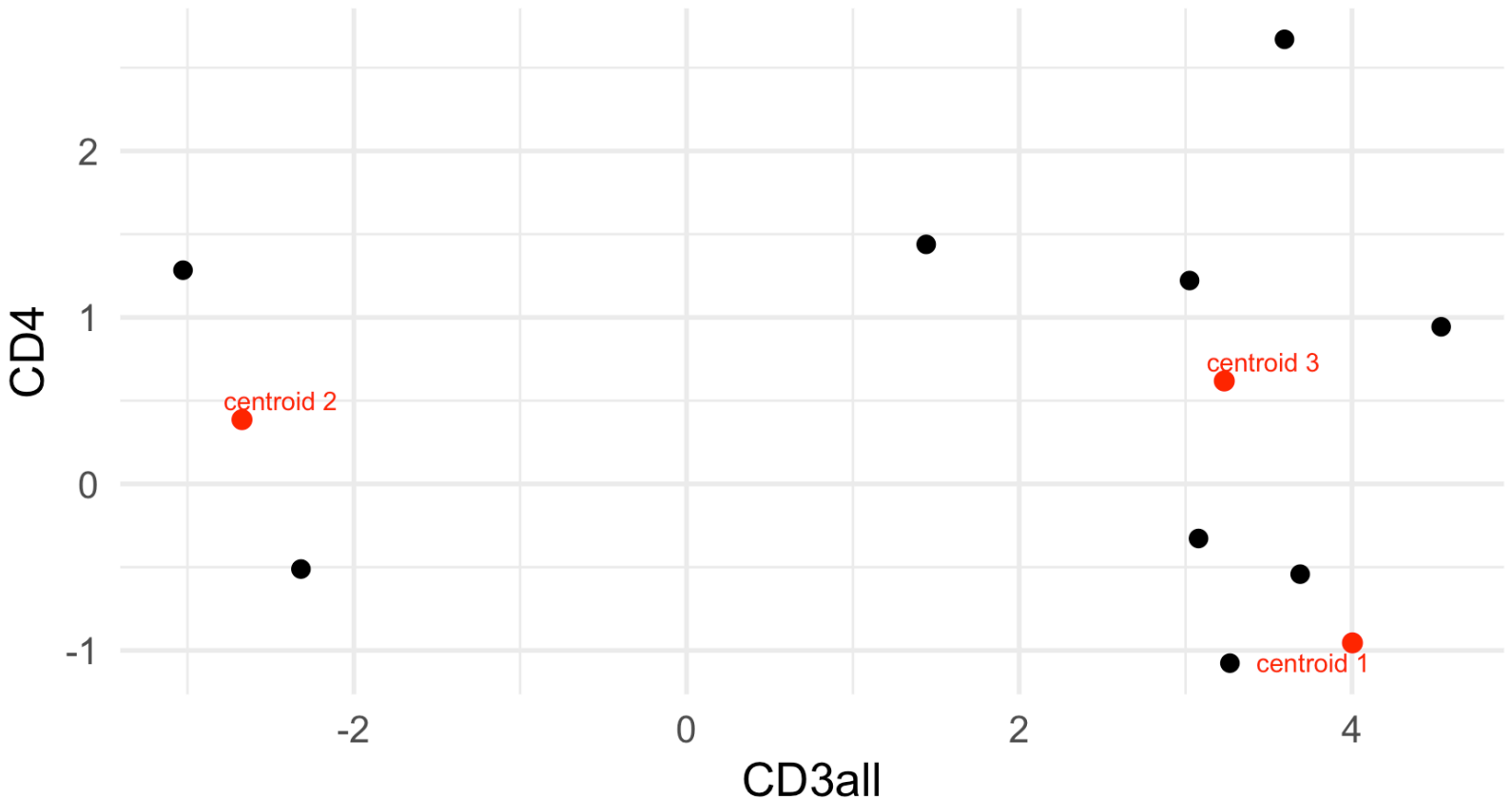
```
 1  2  3  4  5  6  7  8  9 10
 1  2  3  3  3  3  3  3  3  2
```



Example: k-means

```
1 # compute new centroids
2 (centroids <- compute_centroids(cd10, labels, k = 3))
```

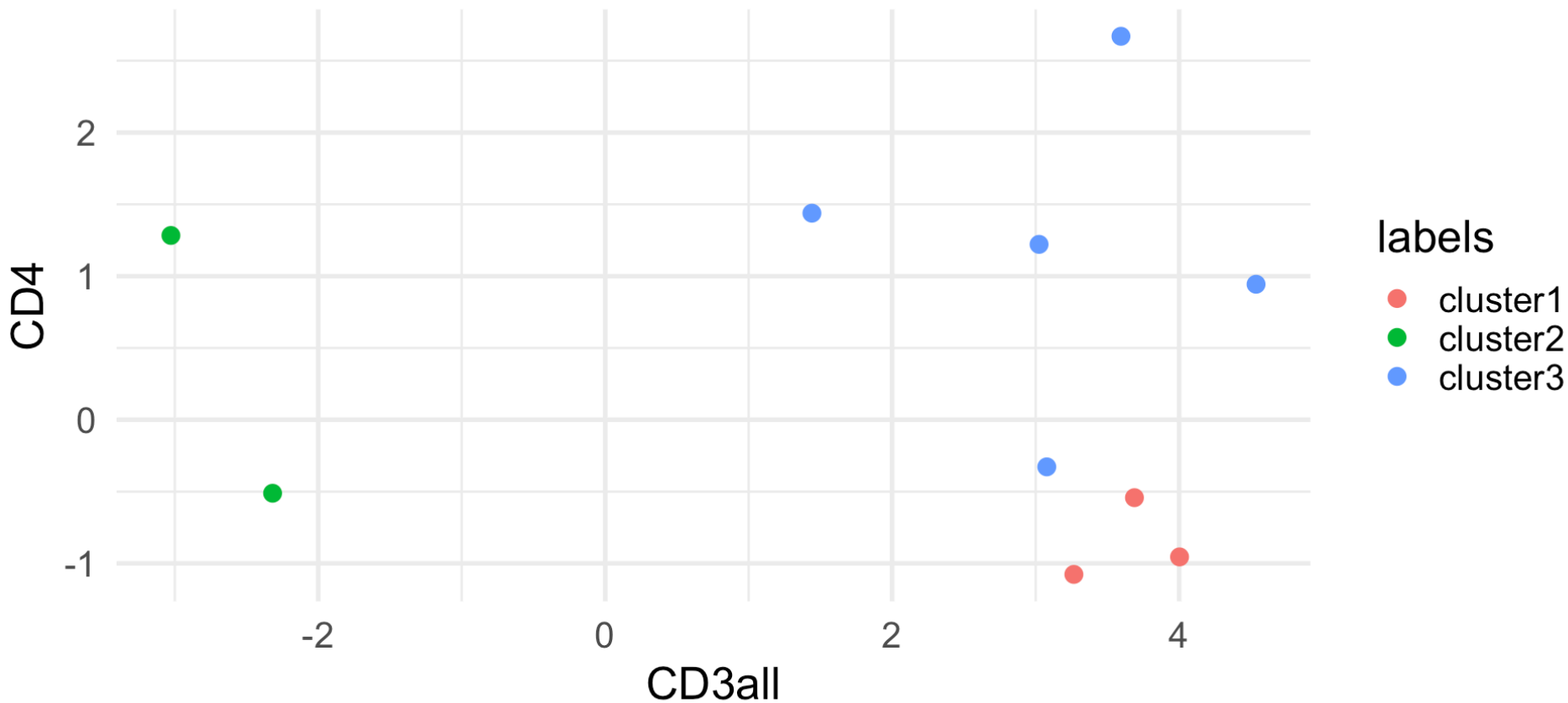
```
       CD3all        CD4
[1,]  4.002830 -0.9547135
[2,] -2.672505  0.3860927
[3,]  3.232513  0.6182287
```



Example: k-means

```
1  # assign observations to new centroids
2  (labels <- assign_centroids(cd10, centroids))
```

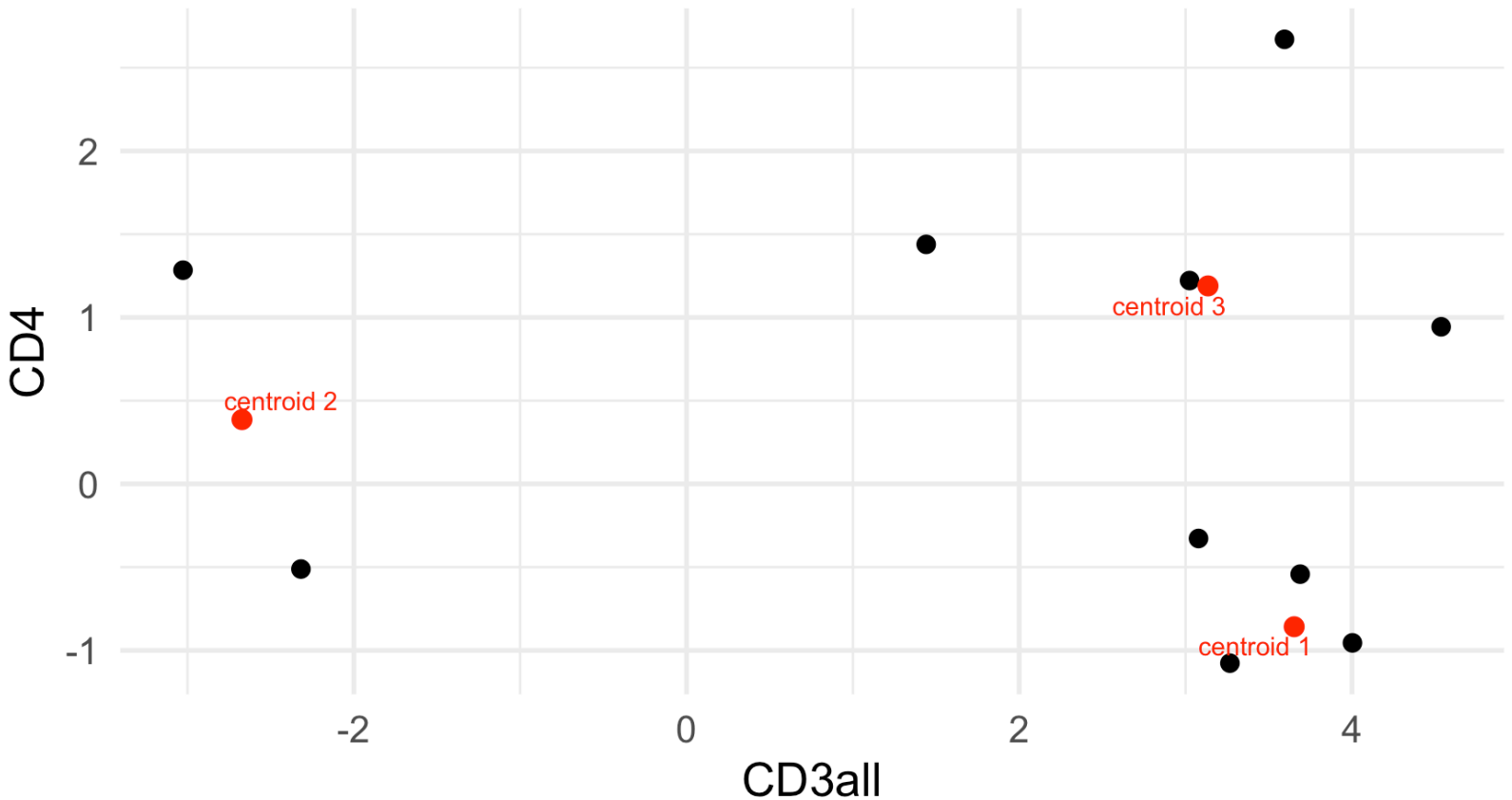
```
 1  2  3  4  5  6  7  8  9 10
 1  2  1  3  1  3  3  3  3  2
```



Example: k-means

```
1 # compute new centroids
2 (centroids <- compute_centroids(cd10, labels, k = 3))
```

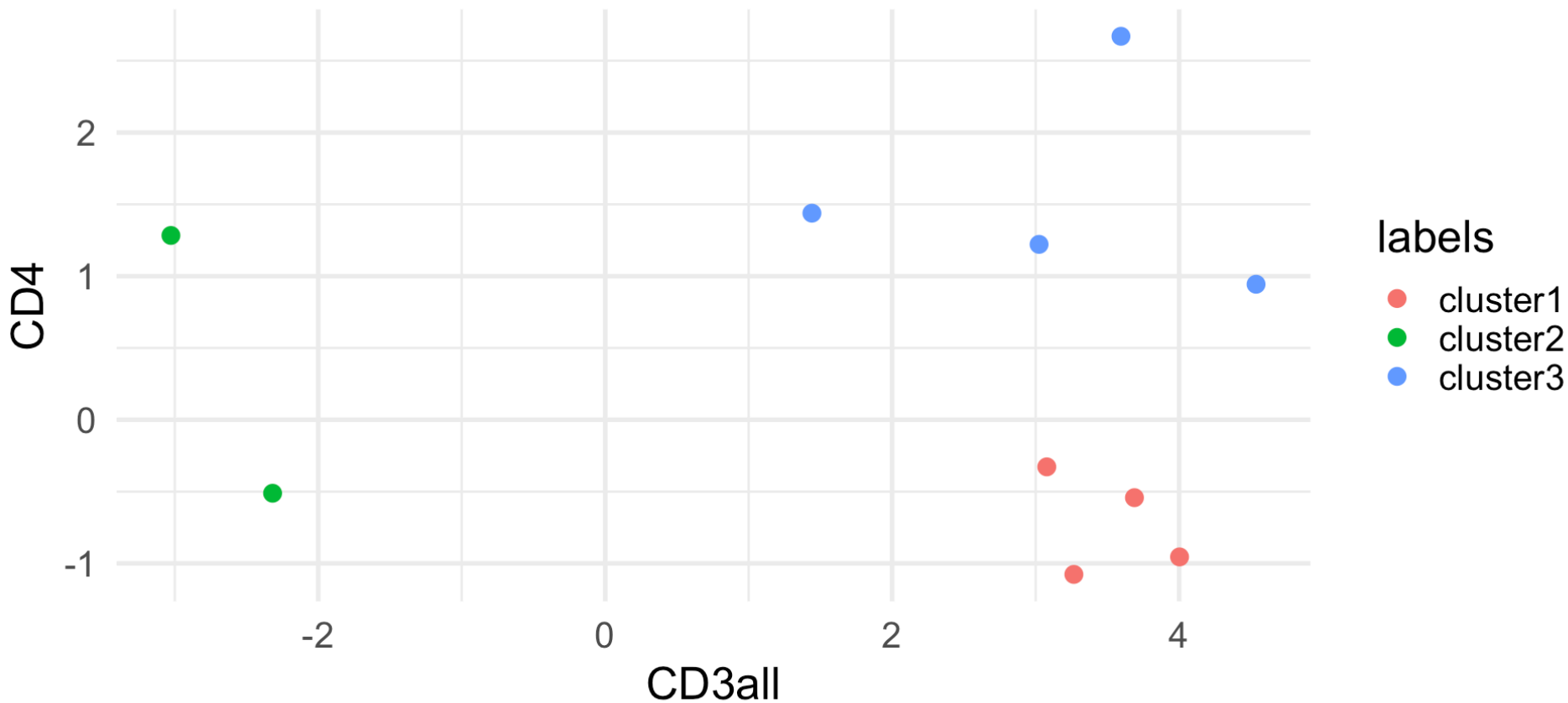
```
        CD3all        CD4
[1,]  3.652583 -0.8577842
[2,] -2.672505  0.3860927
[3,]  3.134535  1.1892479
```



Example: k-means

```
1  # assign observations to new centroids
2  (labels <- assign_centroids(cd10, centroids))
```

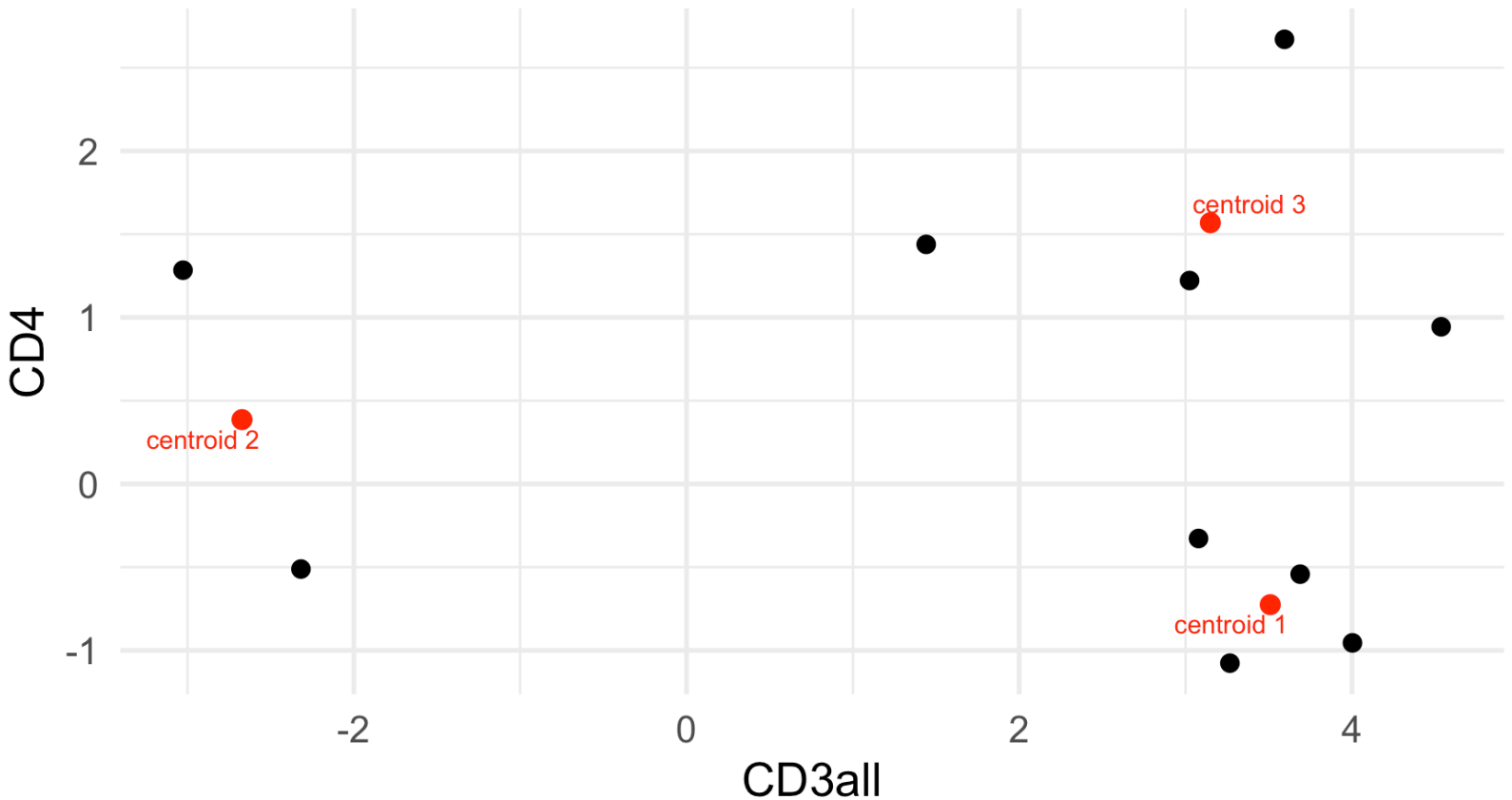
```
 1  2  3  4  5  6  7  8  9 10
 1  2  1  3  1  3  1  3  3  2
```



Example: k-means

```
1 # compute new centroids
2 (centroids <- compute_centroids(cd10, labels, k = 3))
```

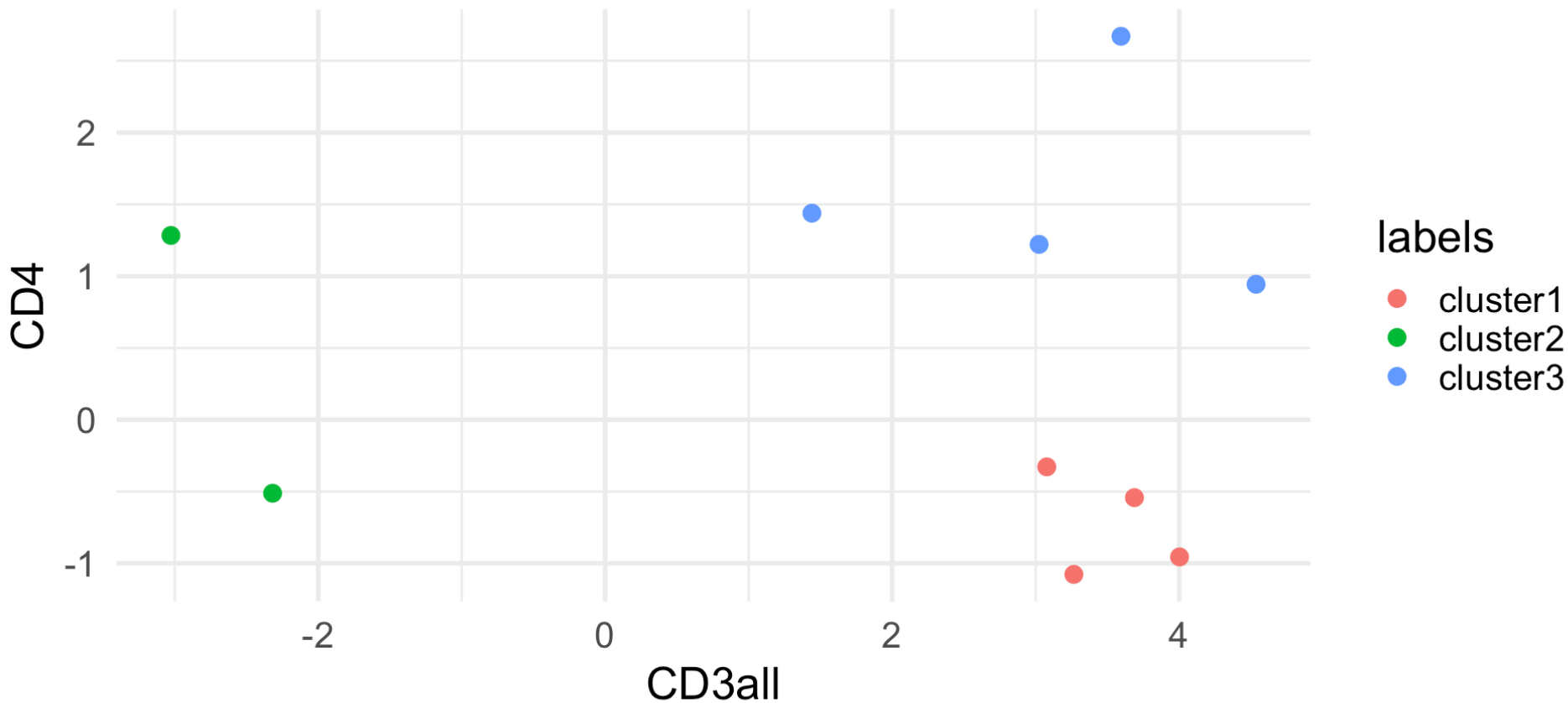
```
        CD3all        CD4
[1,]  3.508820 -0.7252677
[2,] -2.672505  0.3860927
[3,]  3.148786  1.5684894
```



Example: k-means

```
1  # assign observations to new centroids
2  (labels <- assign_centroids(cd10, centroids))
```

```
 1  2  3  4  5  6  7  8  9 10
 1  2  1  3  1  3  1  3  3  2
```



Randomness of k-means

The k-means algorithm is not deterministic, because of the random initialization of the centroids.

Hence, every time you run the algorithm, you may get different results. It is good practice to run the algorithm multiple times and select the solution that minimizes the within-group sum of squares.

K-means in R

```r
set.seed(2258)
(km <- kmeans(cd10, centers = 3, nstart = 10))
```

```
K-means clustering with 3 clusters of sizes 5, 3, 2

Cluster means:
     CD3all        CD4
1  3.714307 -0.3914656
2  2.686297  1.7767384
3 -2.672505  0.3860927

Clustering vector:
 1  2  3  4  5  6  7  8  9 10
 1  3  1  2  1  2  1  1  2  3

Within cluster sum of squares by cluster:
[1] 3.961780 3.711665 1.862013
```

The curse of dimensionality

In this simple example, we have only two variables and we can easily compute the distance between observations in a two-dimensional space.

When dealing with high-dimensional data, the curse of dimensionality makes it less effective to cluster points based on their Euclidean distance.

Hence, it is often useful to perform dimensionality reduction, and perform k-means clustering in the reduced space (e.g, by using the first few principal components).

Evaluating clustering results

The main problem with any clustering algorithm is that, directly or indirectly, we need to set the number of clusters.

As with any algorithm, if we ask for clusters, we will get clusters. But how do we evaluate if the results reflect the underlying structure of the data?

Evaluating clustering results is an active area of research in statistics. Here, we will discuss only two simple metrics that may be useful to compare values of k or to get an overall assessment of the clustering performance.

The silhouette width

The **silhouette width** of observation i is defined as

$$sil(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \in [-1, 1],$$

where

- $a(i)$ is the average distance between i and all other observations in the same cluster;
- $b(i)$ is the average distance between i and all the observations of the closest cluster to which i does not belong.

The silhouette width

A high positive value of silhouette indicates that clusters are well separated and well cohesive.

We can use the silhouette to *choose the number of clusters*, by selecting the value of K that maximizes the silhouette index.

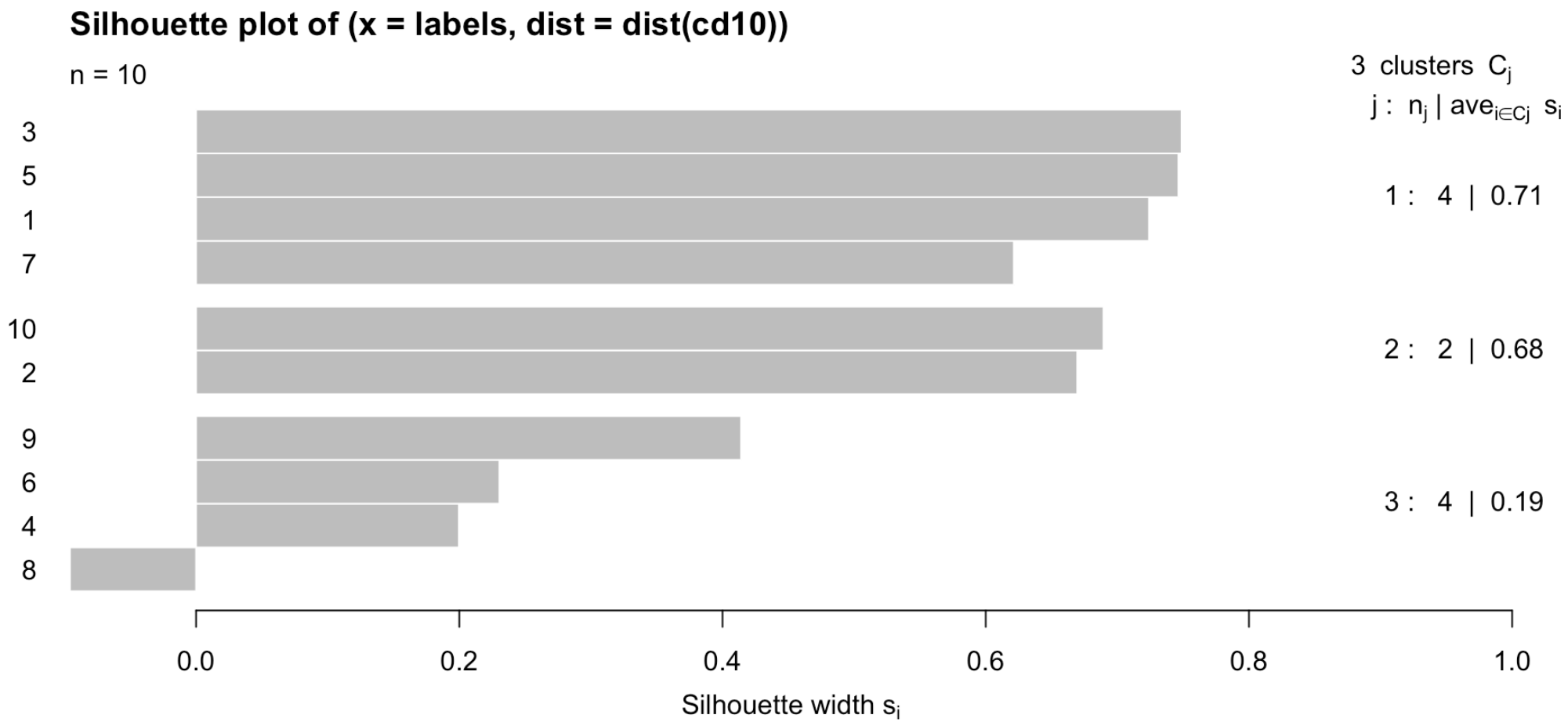
We can look at the average silhouette width for each cluster, and the average silhouette width across all clusters.

Example

```r
library(cluster)
(sil <- silhouette(labels, dist(cd10)))
```

```
      cluster neighbor  sil_width
 [1,]       1        3  0.7240183
 [2,]       2        1  0.6694881
 [3,]       1        3  0.7487952
 [4,]       3        1  0.1996447
 [5,]       1        3  0.7463943
 [6,]       3        1  0.2304958
 [7,]       1        3  0.6213529
 [8,]       3        1 -0.0958473
 [9,]       3        1  0.4140707
[10,]       2        3  0.6894605
attr(,"Ordered")
[1] FALSE
attr(,"call")
```

Example


Silhouette plot of (x = labels, dist = dist(cd10))
n = 10
3 clusters C_j
j : n_j | $ave_{i \in Cj}$ s_i
3
5
1
7
10
2
9
6
4
8
1 : 4 | 0.71
2 : 2 | 0.68
3 : 4 | 0.19
0.0
0.2
0.4
0.6
0.8
1.0
Silhouette width s_i
Average silhouette width : 0.49

The Rand Index

When we want to compare two partitions, we can use the *Rand index*.

This could be useful, for instance, to:

- Compare two different clustering algorithmns;
- Compare the results of a clustering algorithm to ground truth labels.

The Rand Index

Given two partitions $R = \{R_1, \ldots, R_p\}$ and $S = \{S_1, \ldots, S_q\}$, the Rand Index is defined as

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}},$$

where

- a is the number of pairs of elements that are in the same cluster in both R and S.
- b is the number of pairs of elements that are in different clusters in both R and S.
- c is the number of pairs of elements that are in the same cluster in R but in different clusters in S.
- d is the number of pairs of elements that are in the same cluster in S but in different clusters in R.

Adjusted Rand Index

The Rand Index ranges between 0 and 1, where 0 indicates complete disagreement and 1 complete agreement.

Obviously, if we compare two partitions, it is likely that the two partitions agree by chance for some pairs of observations.

There is an “adjusted” version of the Rand Index that takes into account this element of randomness and that is equal to 0 when the two partitions do not agree more than expected by chance between two random partitions.

Example

Let’s compare our manual k-means solution to the one obtained by the kmeans function in R.

```
cbind(labels, km$cluster)
```

```
   labels
1       1 1
2       2 3
3       1 1
4       3 2
5       1 1
6       3 2
7       1 1
8       3 1
9       3 2
10      2 3
```

```
library(mclust)
adjustedRandIndex(labels, km$cluster)
```

```
[1] 0.6298472
```

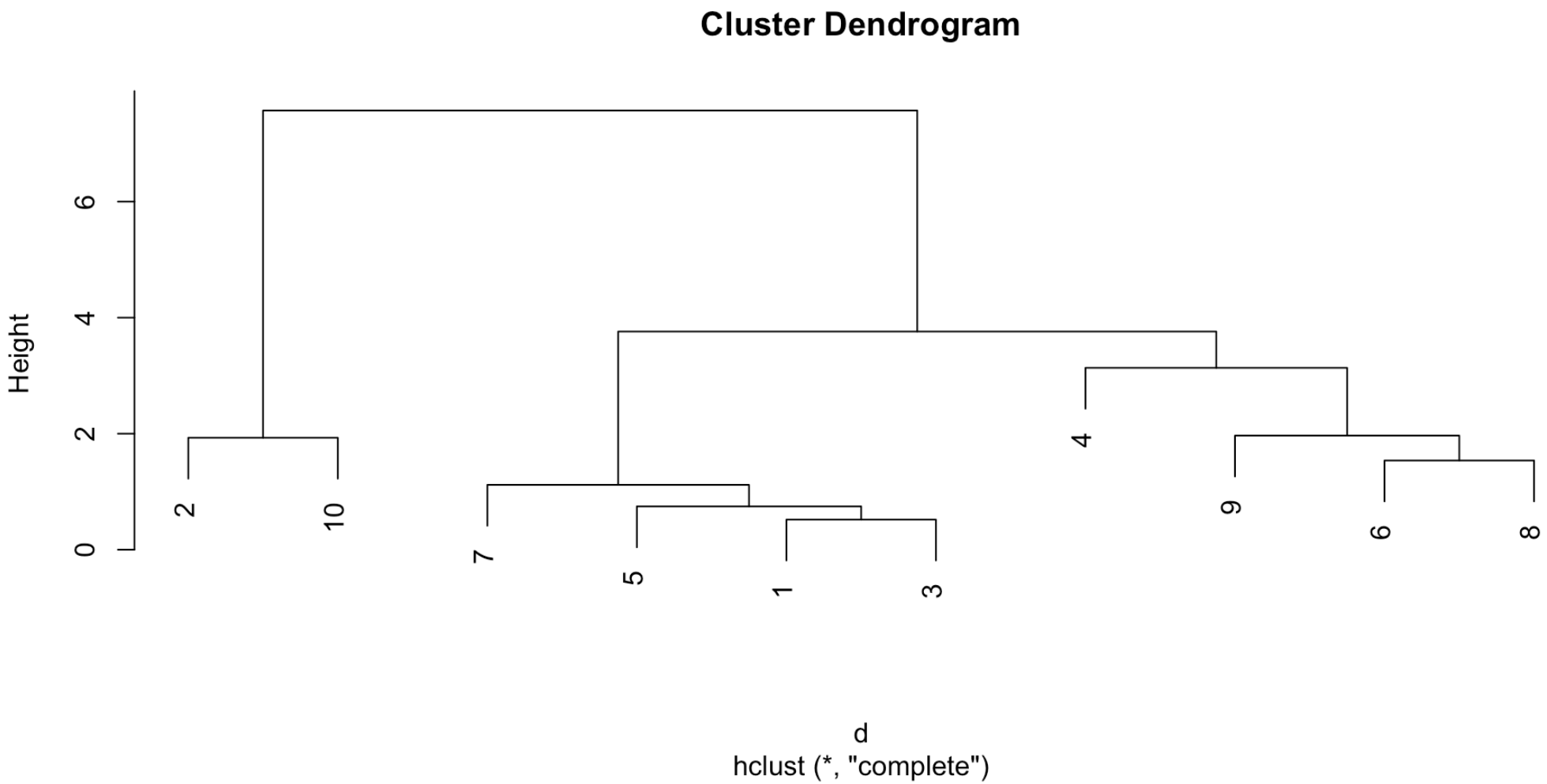
Hierarchical clustering

Hierarchical clustering

A different approach to clustering is to start from the observations and iteratively merge the closest ones into larger and larger groups, until all observations belong to the same group.

This allows us to avoid pre-specifying the number of clusters, and the result that we will get is not a single partition, but a *hierarchy* of partitions, that we can visualize with a *dendrogram*.

Example


Cluster Dendrogram
Height
0
2
4
6
2
10
7
5
1
3
4
9
6
8
d
hclust (*, "complete")

Distance between groups

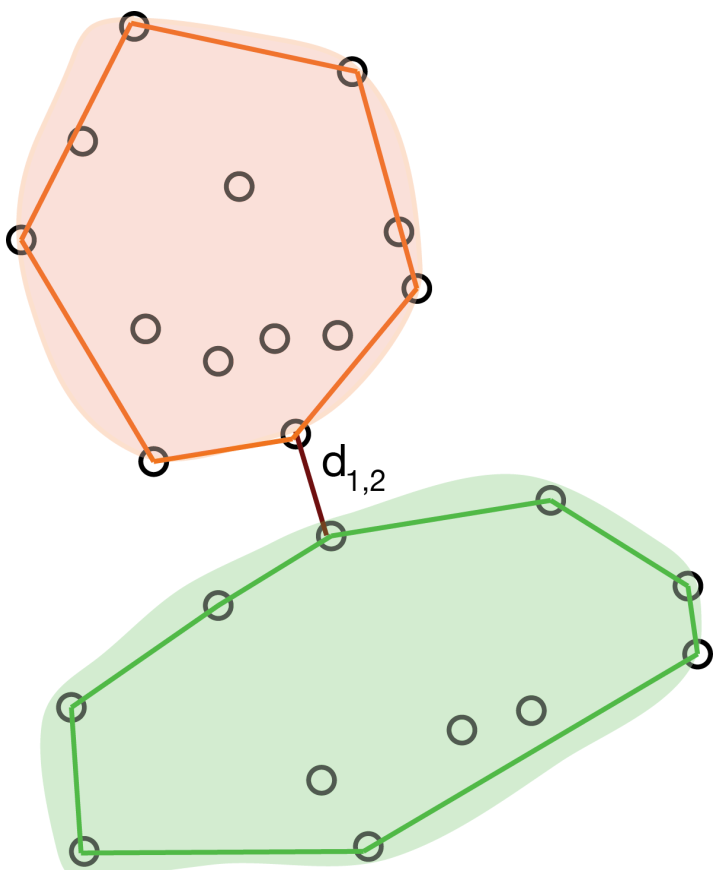
While intuitive, this method requires the definition of a distance between groups of observations, in order to decide which groups to merge at each step.

In fact, you can imagine several ways to define such distance:

- the distance between the closest observations in the two groups;
- the distance between the farthest observations in the two groups;
- the average distance between all pairs of observations in the two groups;
- the distance between the centroids of the two groups.

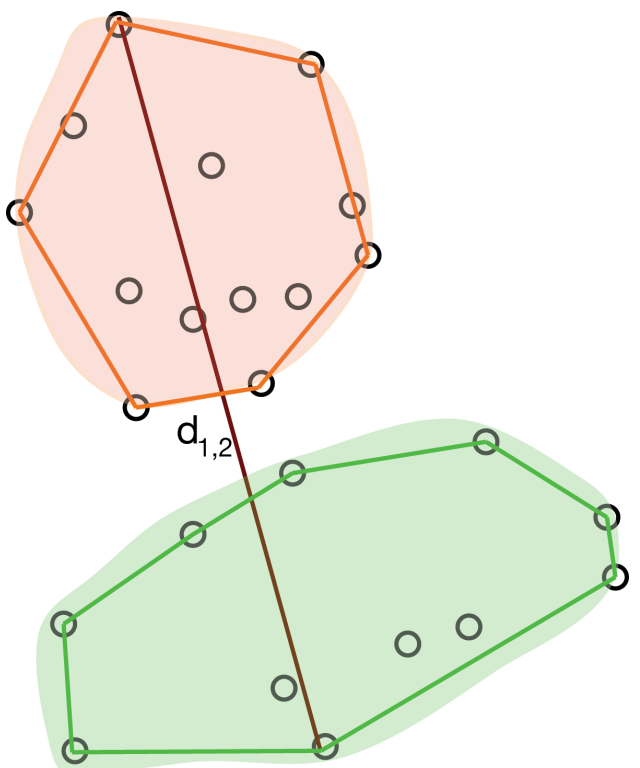
Single linkage

The **single linkage** method defines the distance between two groups as the distance between the closest observations in the two groups.



Complete linkage

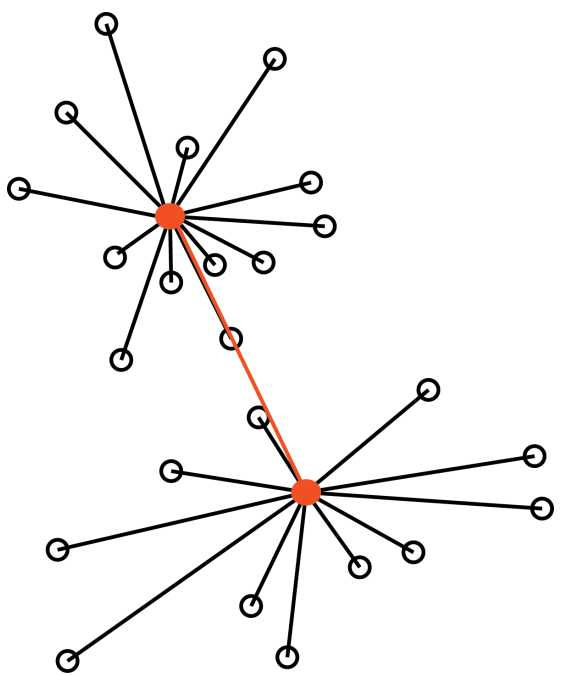
The **complete linkage** method defines the distance between two groups as the distance between the farthest observations in the two groups.



Complete Linkage

The Ward method

The **Ward method** defines the distance between two groups as the increase in the within-group sum of squares that would result from merging the two groups (sum of squared distances between data points and their cluster centroid).



Ward Method

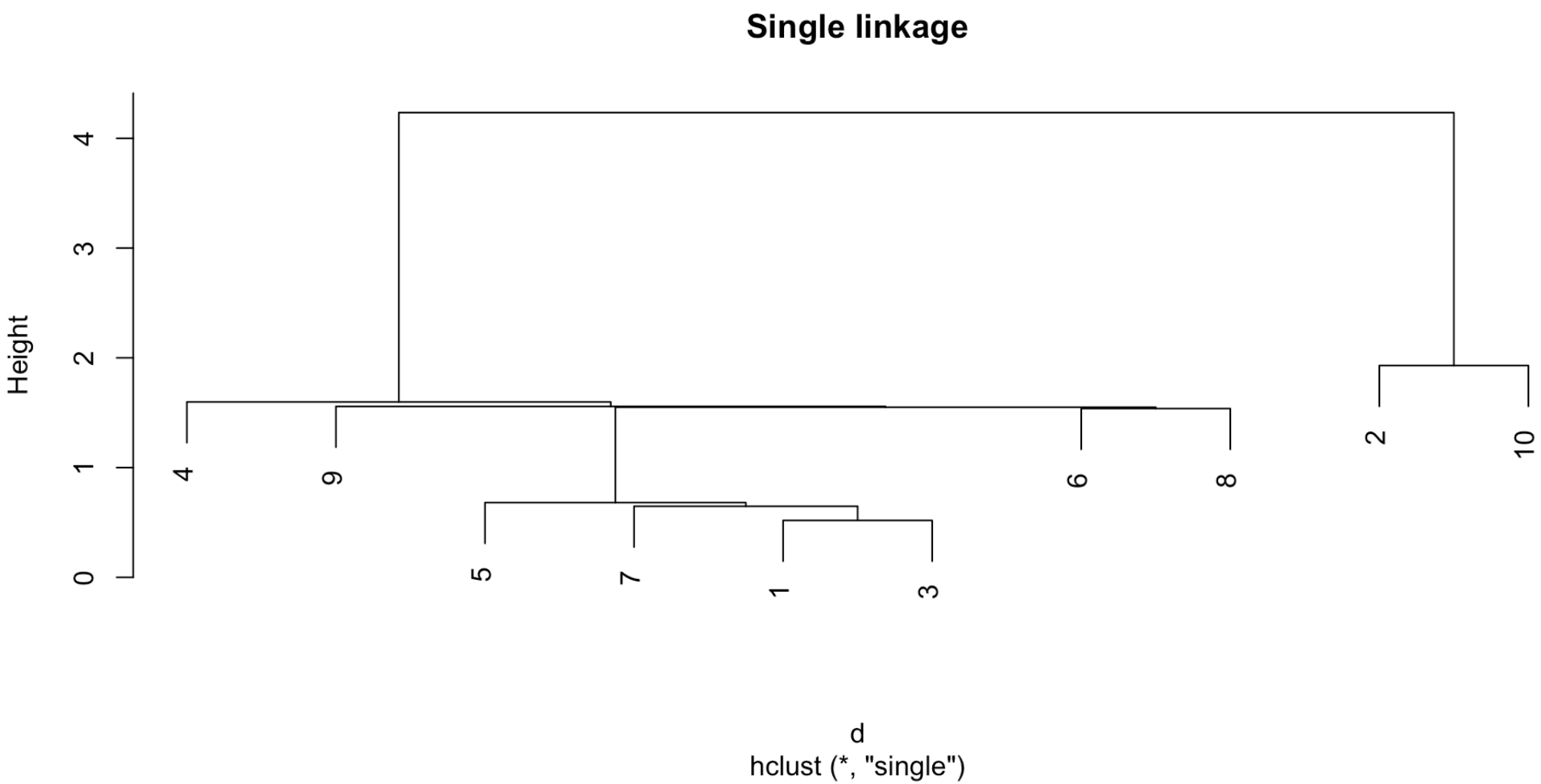
Which method to choose?

The single linkage tend to create long clusters that look like long strings of points, while the complete linkage tends to create compact clusters.

The Ward method gives often good results in practice, but may result in smaller cluster sizes.

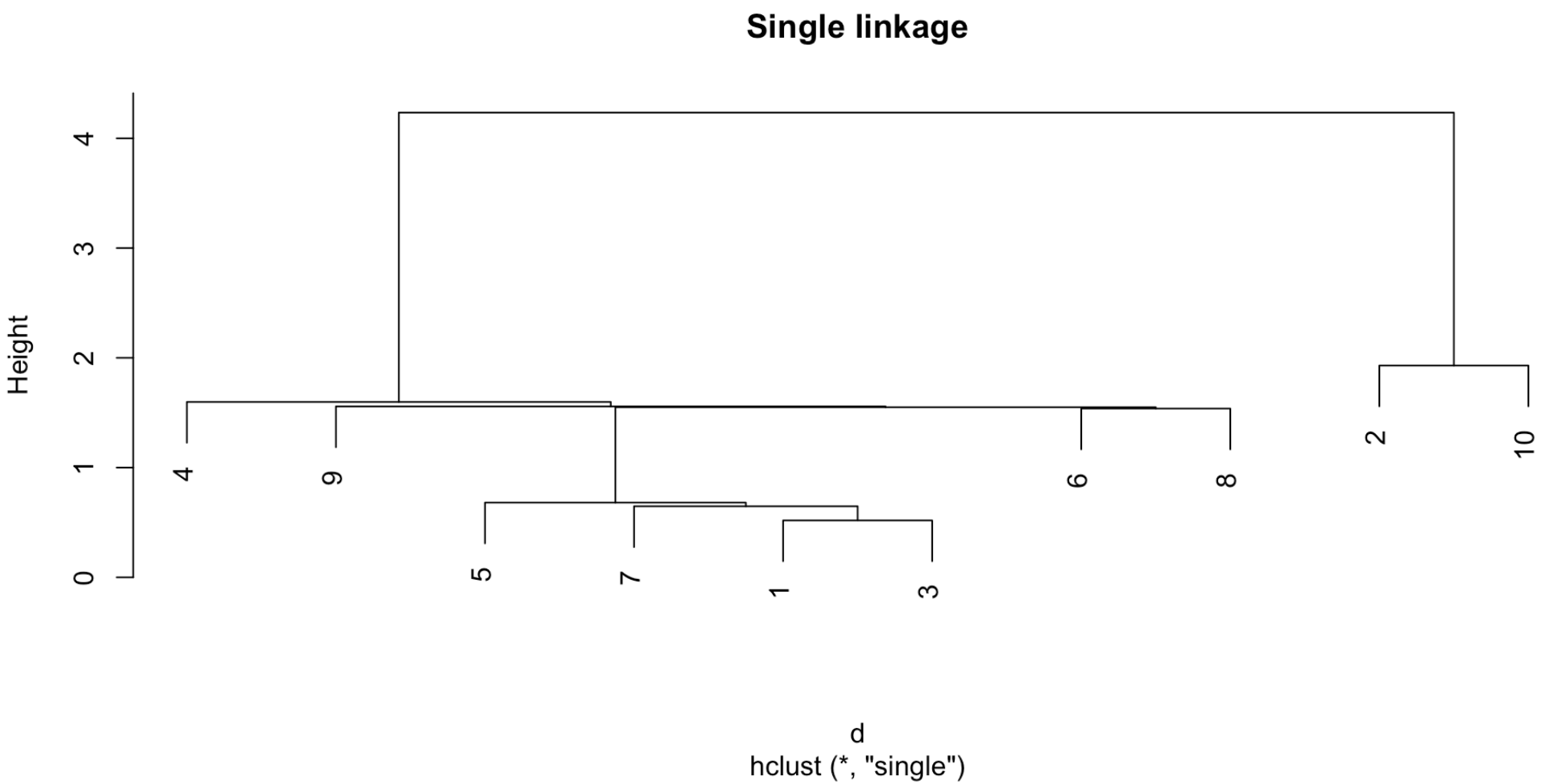
Example

```
1 d <- dist(cd10)
2 hclust(d, method = "single") |> plot(main = "Single linkage")
```



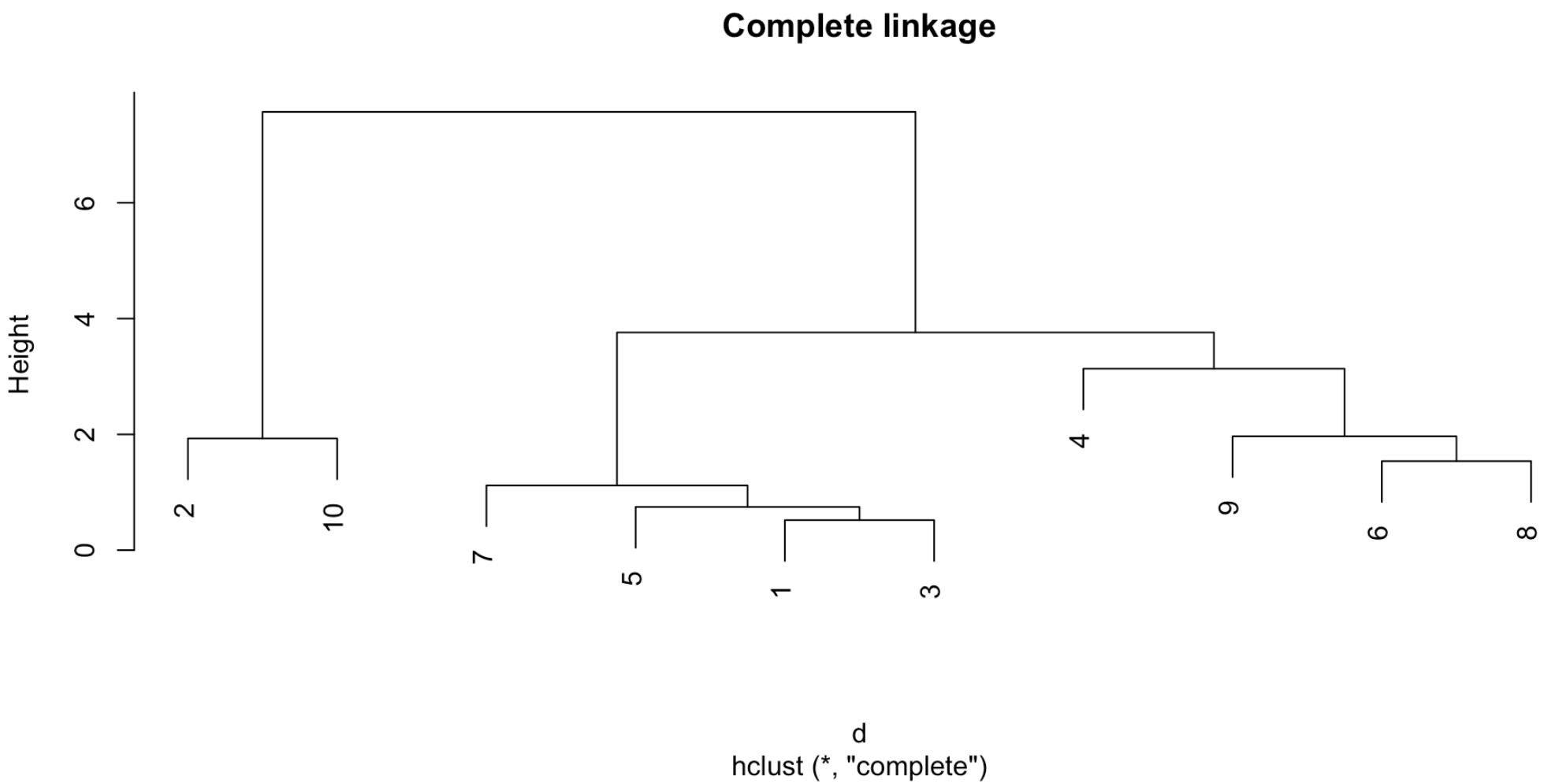
Example

```
1 d <- dist(cd10)
2 hclust(d, method = "single") |> plot(main = "Single linkage")
```



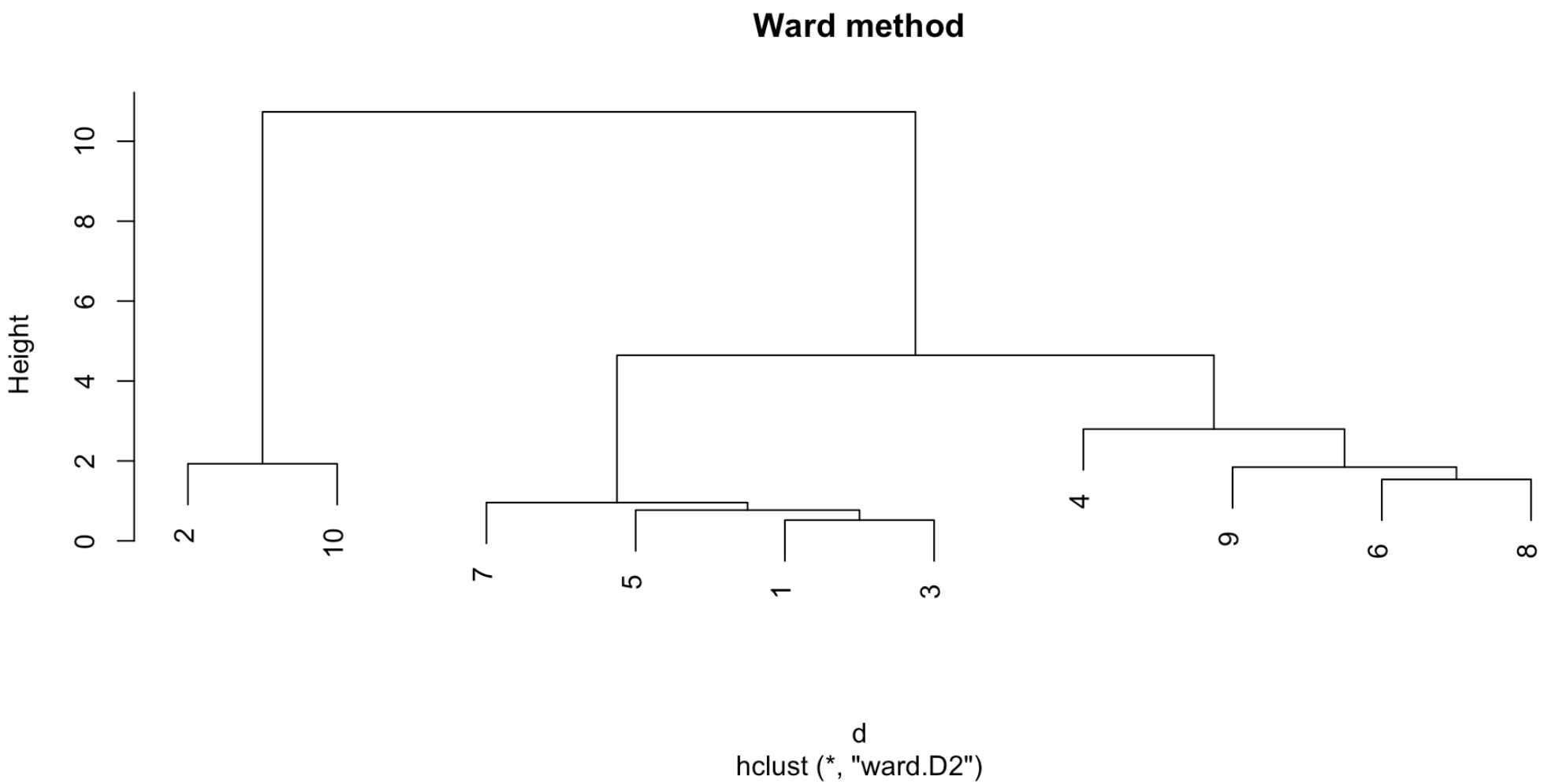
Example

```
1 hclust(d, method = "complete") |> plot(main = "Complete linkage")
```



Example

```
1 hclust(d, method = "ward.D2") |> plot(main = "Ward method")
```



Cutting the dendrogram

While the dendrogram gives us a “global view” of the clustering, it may be practical to have a proper data partition.

In this case, we can “cut” the dendrogram at a certain height.

Typically, we want to cut the dendrogram at a height that corresponds to a “big jump” in the distance between groups, but this is not always easy to identify.

Partitional vs hierarchical clustering

Hierarchical clustering returns a hierarchy of groups, which one could argue is a more informative result than a single partition.

However, hierarchical clustering is more computationally expensive than partitional methods, and it does not scale well to large datasets.

Finally, great care needs to be taken in interpreting hierarchical clustering results:

- we are imposing a hierarchy, which may not be present in the data;
- the left-right order of the branches is arbitrary and may be tricky to interpret.

Can we identify CD4+ T cells?