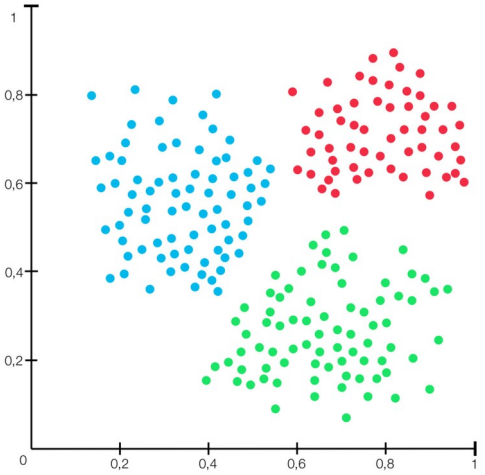
R-course:
Machine Learning using R

K-means clustering

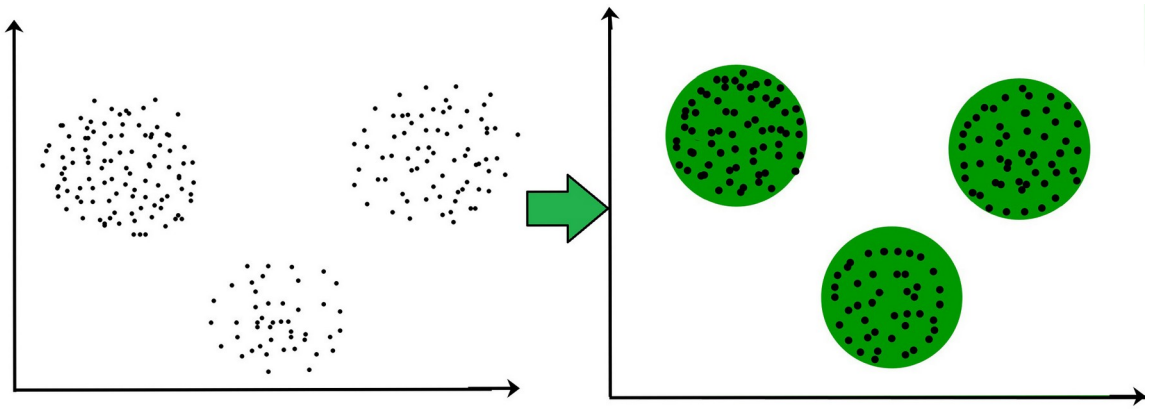


Yannick Rothacher

Zürich, 2021

Clustering

- Clustering is the process of dividing data points into specific groups
- The idea is that members of the same group/cluster are more similar to each other than they are to members of other clusters
- We need some measure of **similarity** or distance in order to do this
  - One example is the **euclidean distance** as a measure of similarity between two points
  - Euclidean distance in n-dimensional space: $dist(o_{1,}o_2)=\sqrt{\sum_{i=1}^{n}(o_{1i}-o_{2i})^2}$

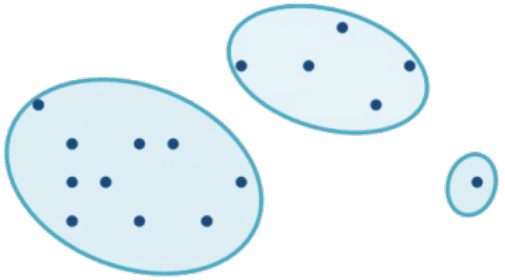


Partitional vs. hierarchical clustering

- There are two types of clustering methods:

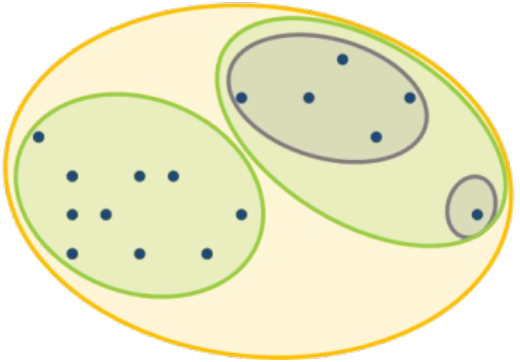
Partitional:

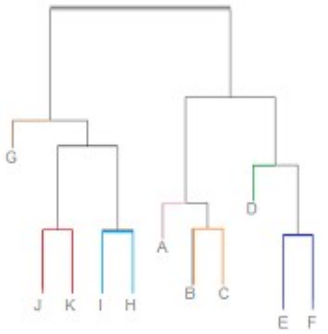
Divide the data into non-overlapping clusters. Each data point is in only one cluster.



Hierarchical:

Divide the data into nested, hierarchical clusters. A data point can be a member of multiple clusters.

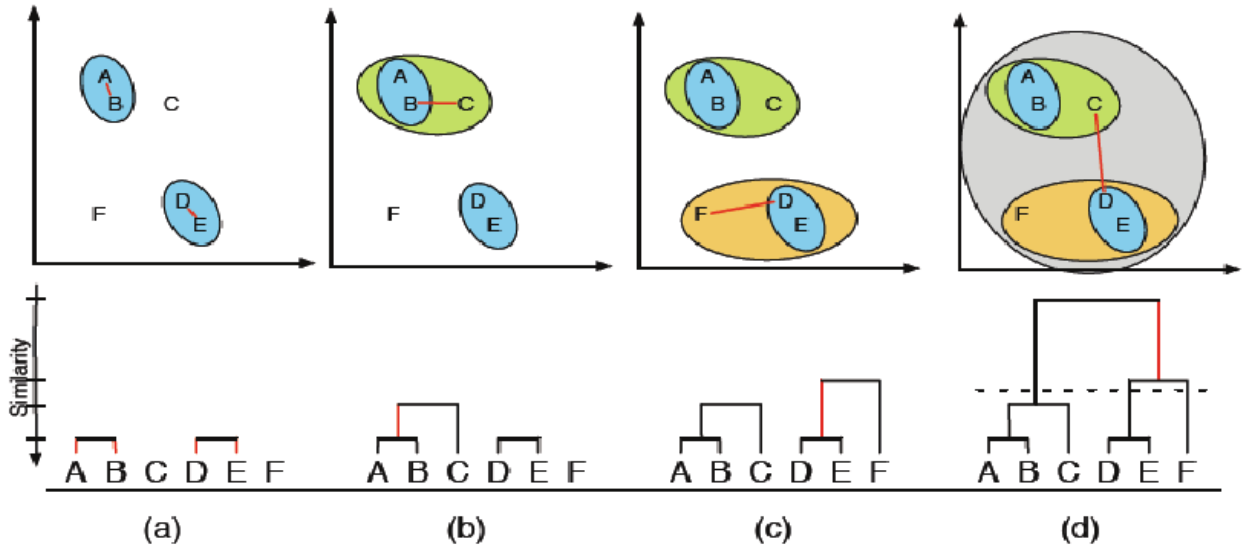




Tree-structure of hierarchical clustering

Recap: Hierarchical clustering

- Hierarchical clustering is performed heuristically either from the bottom-up (**agglomerative**) or top-down (**divisive**)
  - In bottom-up clustering one starts with each data point in its own cluster and tries to merge the best pair of clusters into a new clusters. This is repeated until only one cluster remains.
  - In top-down clustering one starts with all data points in one big cluster and tries to find the best separation of a cluster into two clusters. This is repeated until every data point forms its own cluster.
- Visualization of agglomerative clustering with a dendogram:

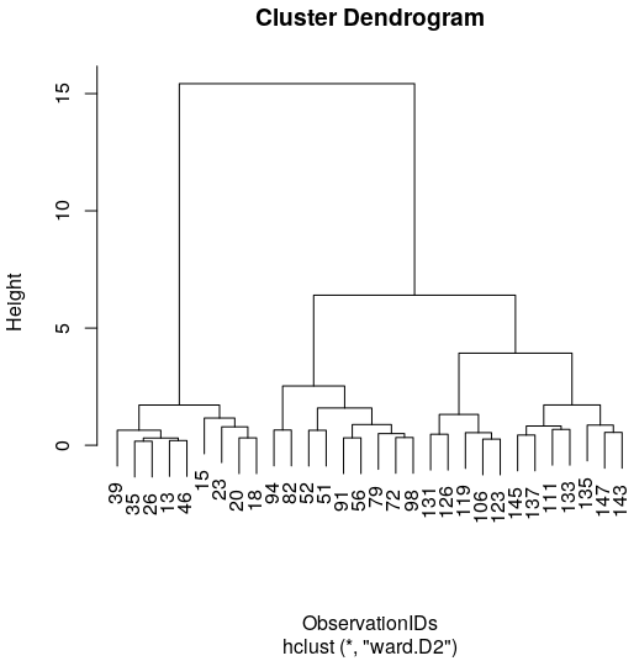


Recap: Hierarchical clustering

- In order for hierarchical clustering to work, we need a way of expressing similarity between different clusters
  - There are different ways to rate the "**distance**" between two clusters (e.g. average distance between point-pairs linking both clusters (**average**)**;** smallest distance between point-pairs linking both clusters (**single-link**))
  - Ward's method tries to minimize the inherent variance of the newly formed clusters
- Hierarchical clustering in R (using Ward's method):

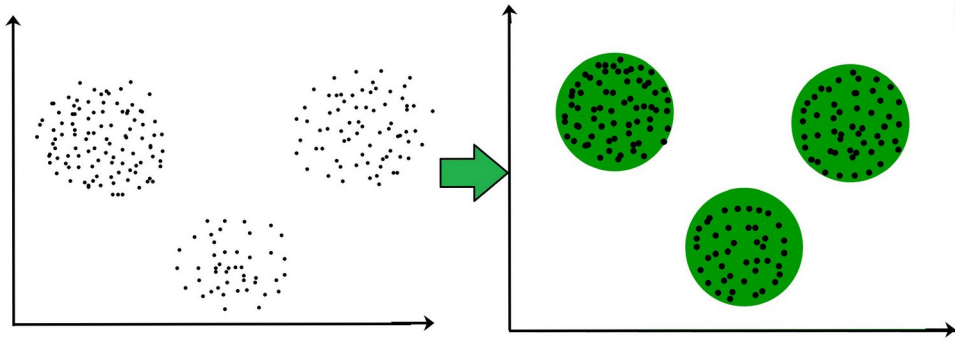
```
# Calculate Euclidean distances:
dist.points <- dist(dat)
# Apply agglomerative clustering:
hc <- hclust(dist.points, method = 'ward.D2')
# Plot the dendogram
plot(hc, xlab = 'ObservationIDs')
# Split data into top two clusters:
cutree(hc, k=2)
```

```
 35 147 145 131  20 111  15 135  72  52  91 119 106  39  23  94 126 133  13
  1   2   2   2   1   2   1   2   2   2   2   2   2   1   1   2   2   2   1
 56 123  98  82  18 143  46  26 137  51  79
  2   2   2   2   1   2   1   1   2   2   2
```



K-means clustering

- K-means clustering is an iterative, **partitional clustering** algorithm
- The goal of **k-means** clustering is to partition the data into **k** clusters with minimal **within-cluster variances**



k = 3

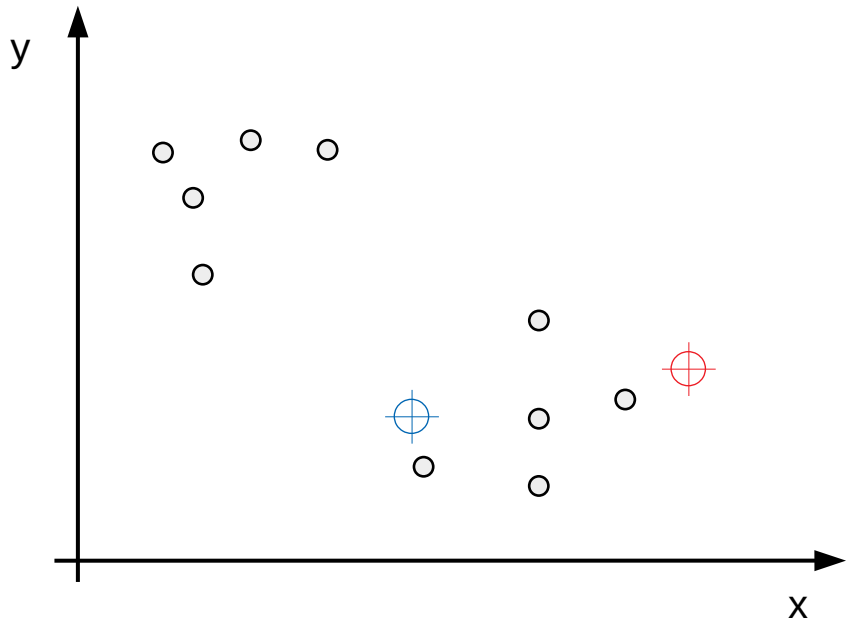
- The within-cluster variance is often based on the Euclidean distances from the cluster center (“centroid”):

$$WCV(C_k) = \frac{1}{|C_k|} \sum_{x_i \in C_k} \|x_i - \bar{x}_k\|^2$$

  - $|C_k|$ is the number of observations in the cluster k and $\bar{x}_k$ is the mean of the cluster k
  - Double bars (|| vector ||) denote the "Euclidean norm" of the vector

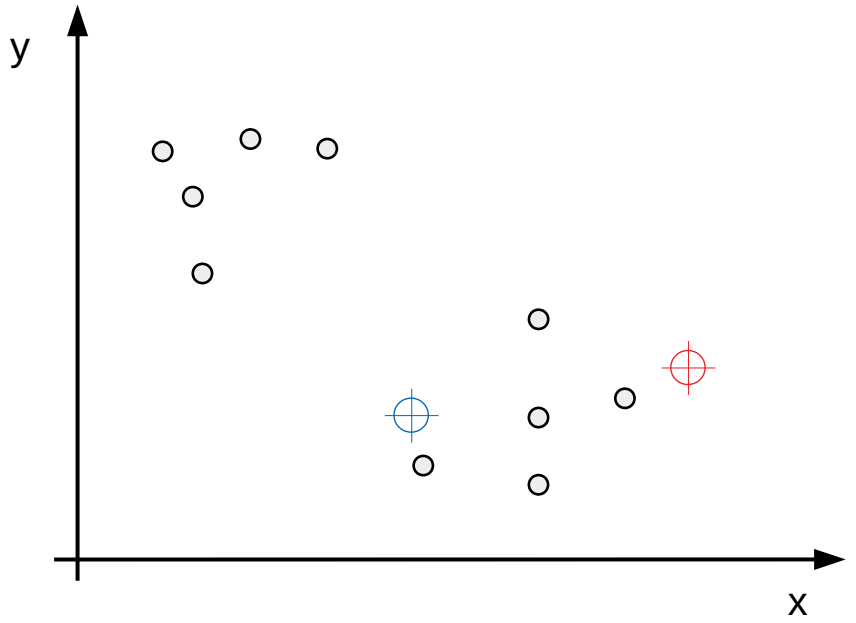
K-means step by step

- The iterative steps of the k-means algorithm are surprisingly simple!
- As a first step, the **number of clusters** has to be defined
  - e.g. **k = 2**
- The algorithm starts with a set of **randomly chosen** k cluster centers:



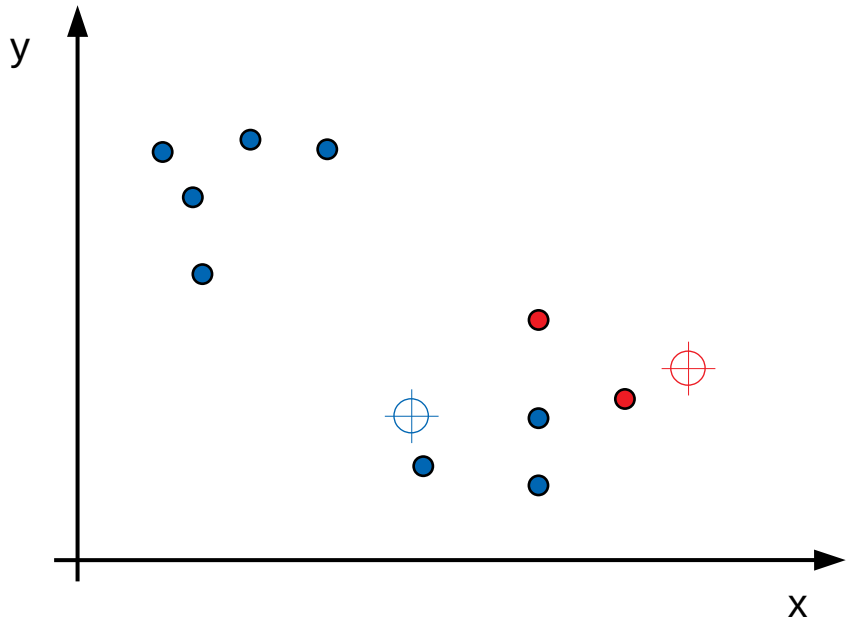
K-means step by step

- The algorithm starts with a set of **randomly chosen** k cluster centers:



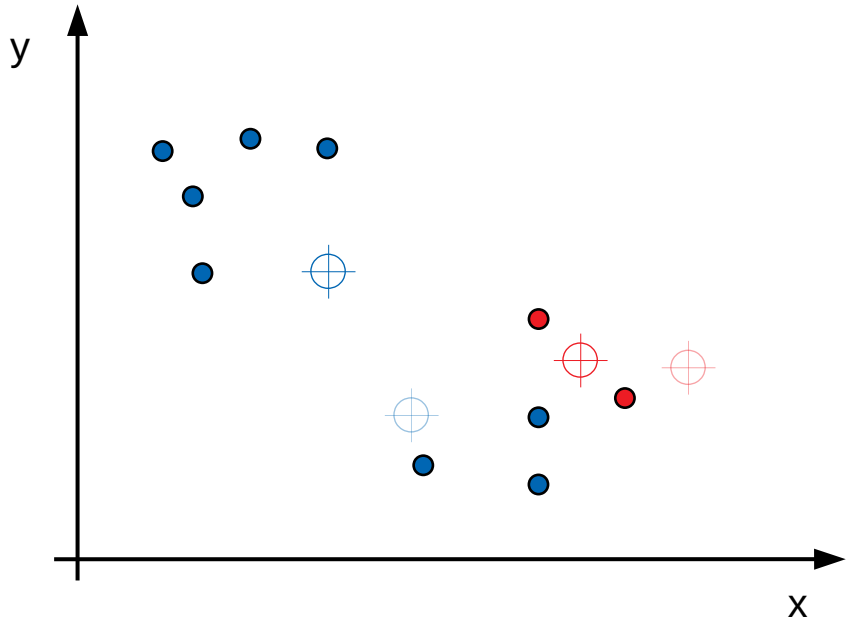
K-means step by step

- Next each data point is assigned to the closest center



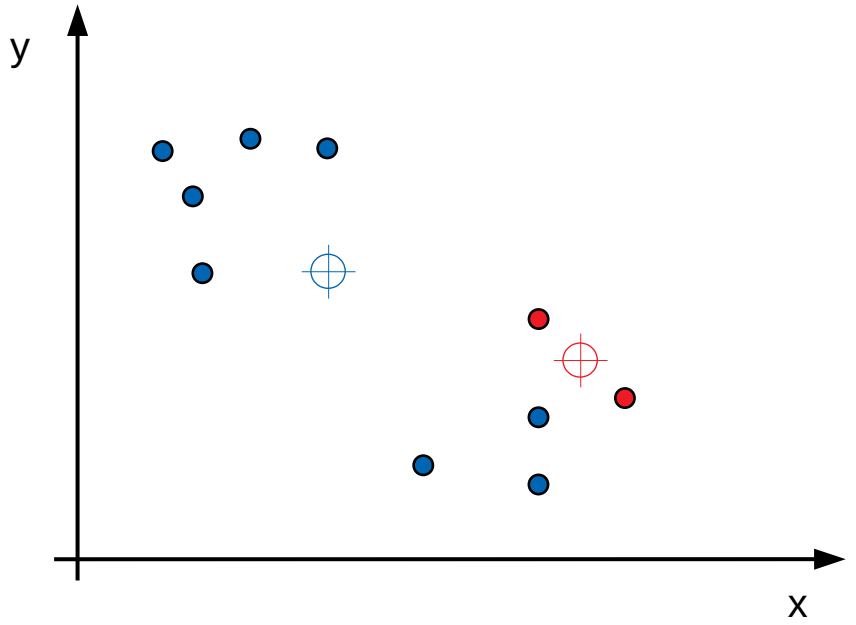
K-means step by step

- Next the two centers are moved to the mean coordinates of the newly assigned data points



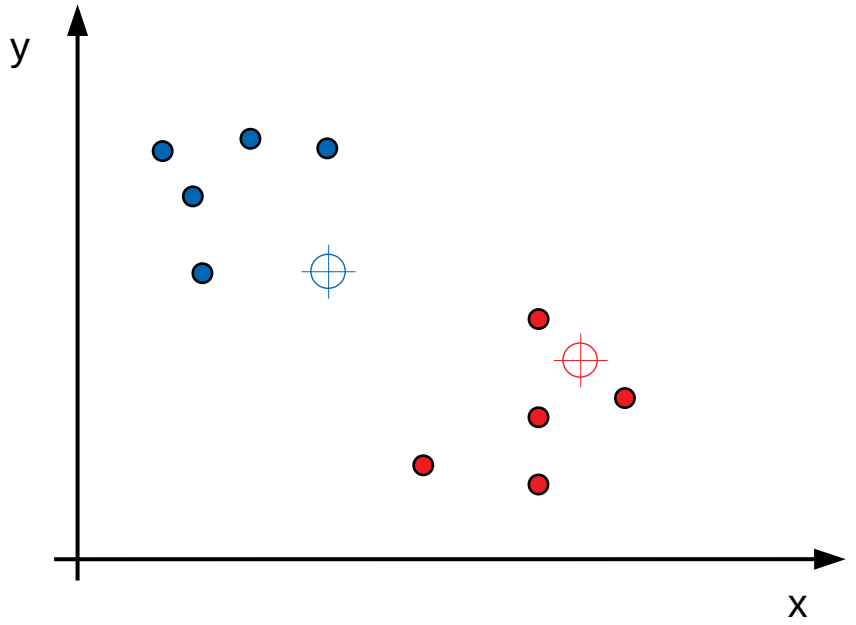
K-means step by step

- Next the two centers are moved to the mean coordinates of the newly assigned data points



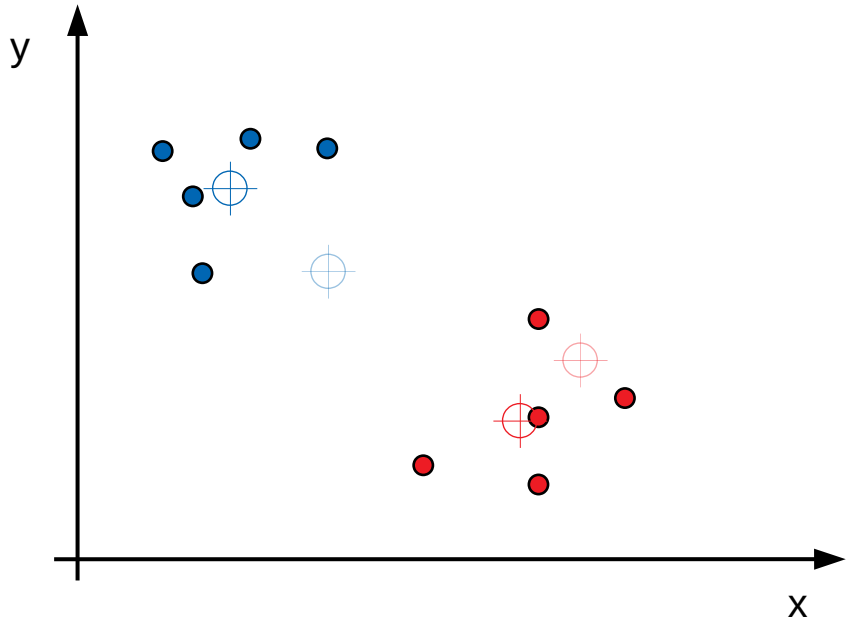
K-means step by step

- Each data point is again assigned to the closest center



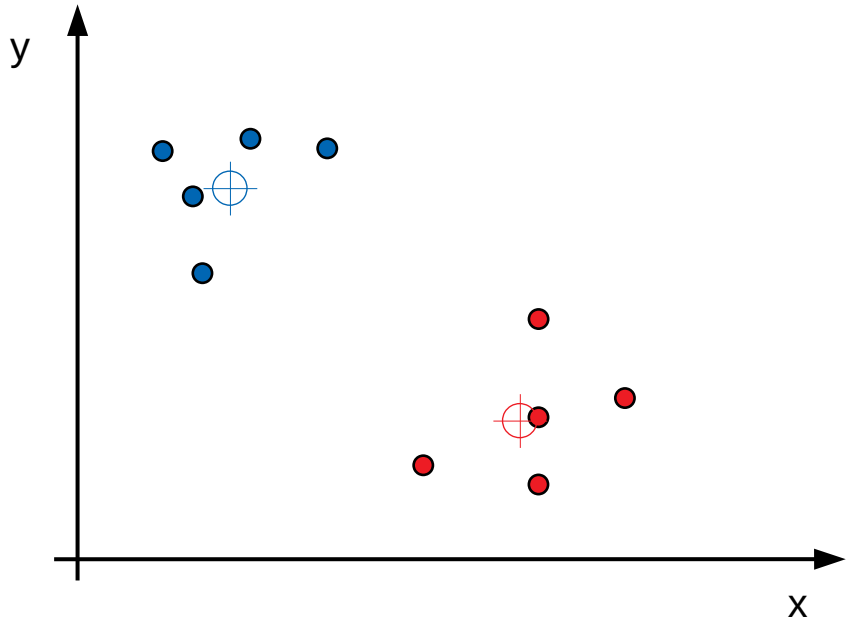
K-means step by step

- The centers are again moved to the mean coordinates of the new clusters



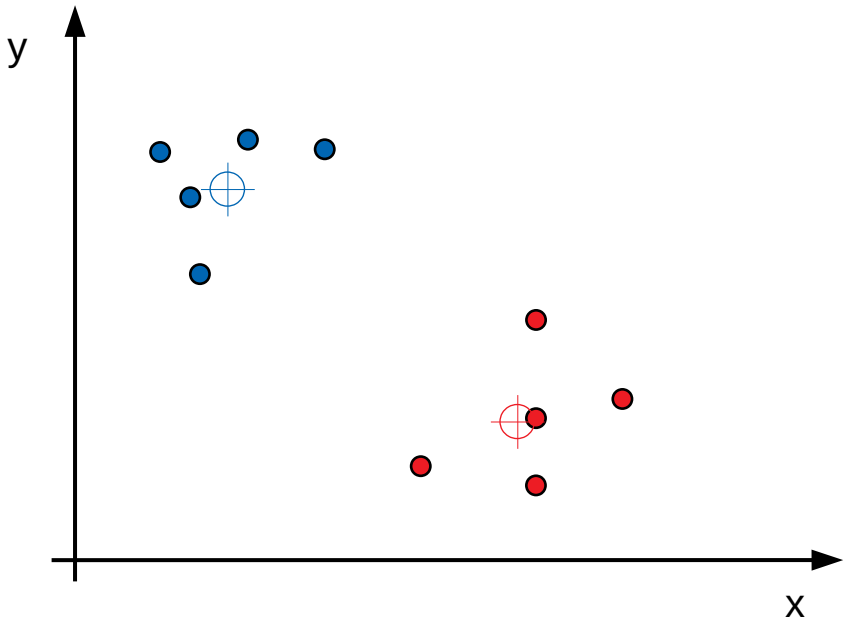
K-means step by step

- The centers are again moved to the mean coordinates of the new clusters



K-means step by step

- This procedure is repeated until the algorithm converges



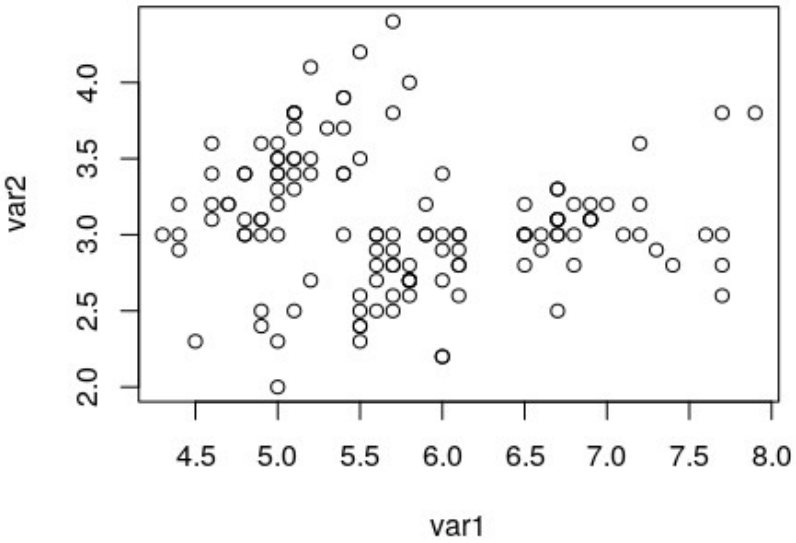
Algorithm 10.1 *K-Means Clustering*

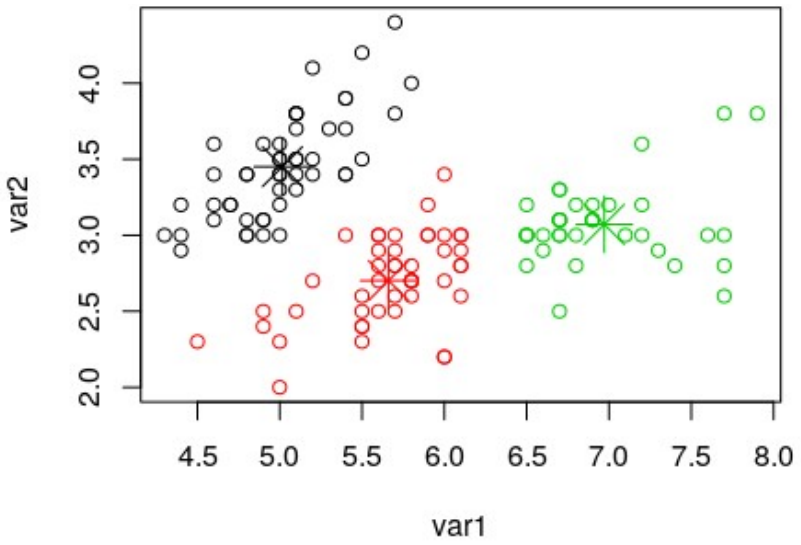
1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:
   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.
   (b) Assign each observation to the cluster whose centroid is closest (where *closest* is defined using Euclidean distance).

K-means clustering in R

```
plot(dat)
# apply k-means algorithm:
set.seed(123)
km <- kmeans(dat, centers=3)
plot(dat, col=km$cluster)
points(km$centers, col=1:3, pch=8, cex=3)

km$cluster  # Show assignment to clusters
[1] 1 2 2 2 3 1 1 2 3 ...
km$centers  # Show coordinates of centers
      var1     var2
1 5.016327 3.451020
2 5.660870 2.702174
3 6.971429 3.071429
```





How many clusters do we need?

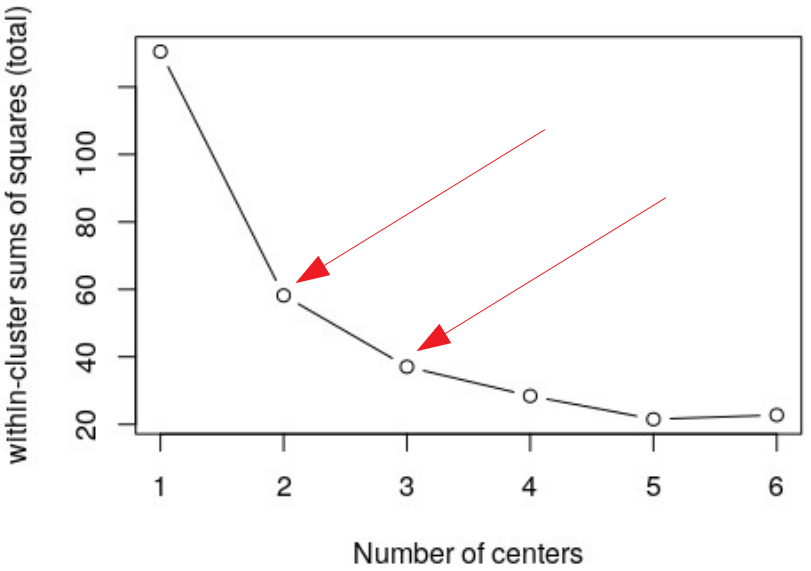
- Check the total of the within-cluster sums of squares for different number of clusters (k)
  - Plot the within-cluster sums of squares vs k, and choose k after the last big drop in variance:

```
wss <- rep(NA, 6)  # initialize

for(i in 1:6){  # Check for up to 6 clusters
  wss[i] <- kmeans(dat, centers = i)$tot.withinss  # Sum of within-clust.SumSquares
}

plot(1:6, wss, type = 'b',
     xlab = 'Number of centers',
     ylab = 'within-cluster sums
     of squares (total)')
```

- Could choose k=2 or maybe k=3

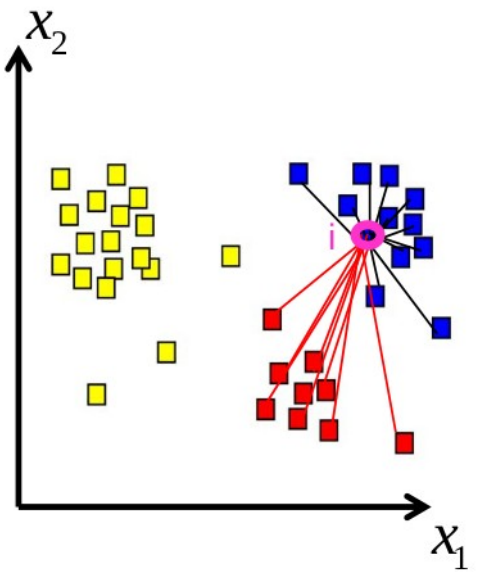


Assessing the quality of the clustering

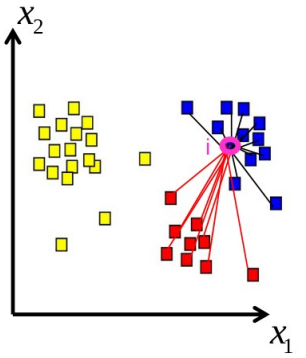
- For a quality-check we can look at the **silhouette** width of each observation
  - Silhouette of observation i :

$$sil_i = \frac{b_i - a_i}{max(b_i, a_i)}$$

  - a_i : Average distance between observation i and all other data points in the same cluster
  - b_i : Average distance between observation i and all data points of the closest cluster ("neighbor cluster")



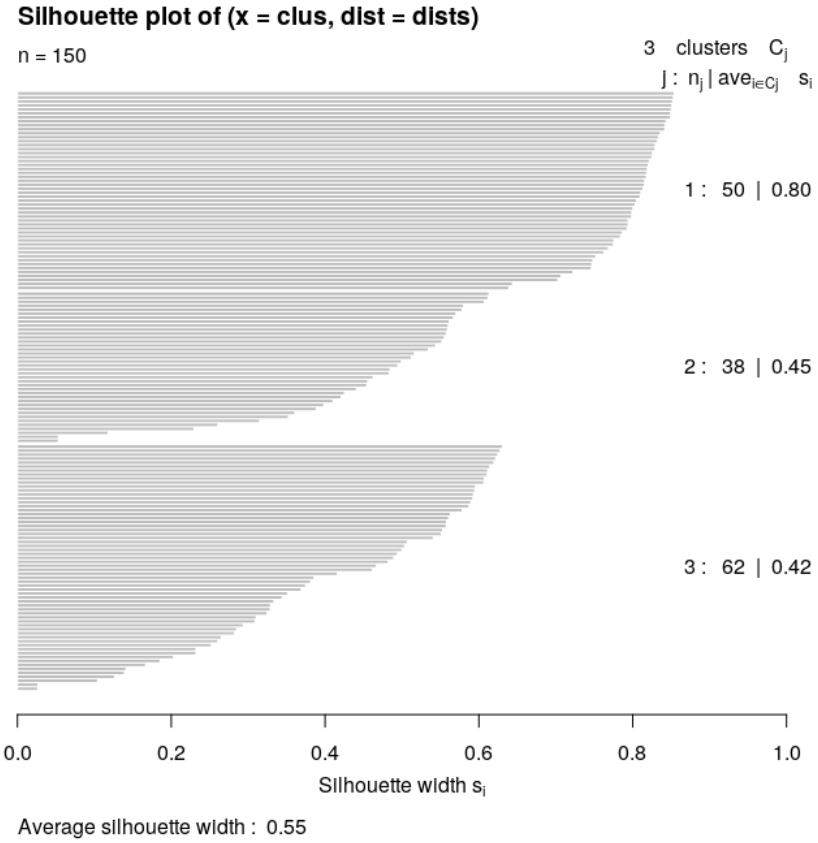
- Silhouette can range from -1 to +1
  - If closest cluster is far away: sil_i goes towards +1 ($a_i < b_i$)
  - If closest cluster is close: sil_i goes towards -1 ($a_i > b_i$)



Silhouette plot

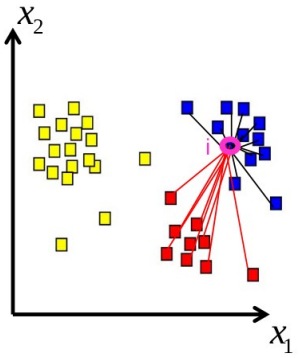
- A silhouette plot visualizes all silhouettes widths:

```
# K-means with iris data
dat <- iris
dat$Species <- NULL # Remove Species column
km <- kmeans(dat, centers = 3)
clus <- km$cluster # Cluster assignments
dists <- dist(dat) # Distance matrix
                   # (Euclidean distance between
                   #  each observation pair)
library(cluster)
plot( silhouette(clus, dists))
```



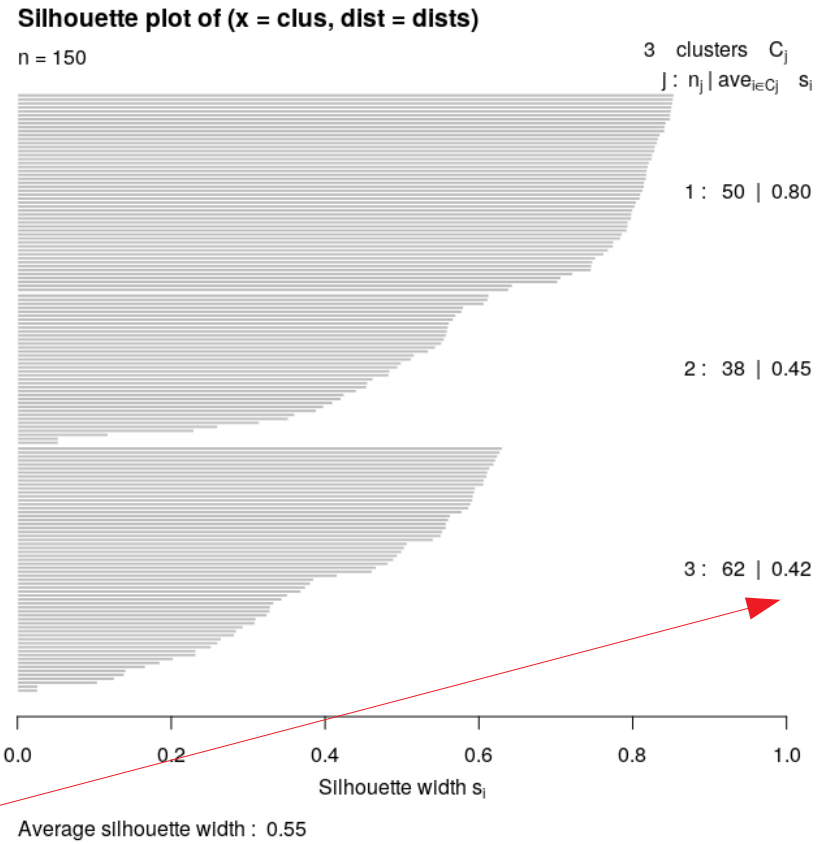
- For each of the three clusters we get the average silhouette width

Silhouette plot

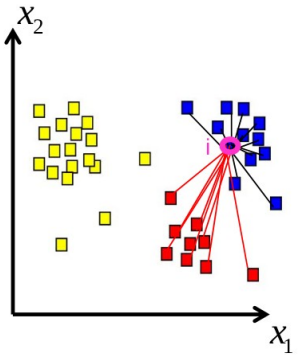


- A silhouette plot visualizes all silhouettes widths:

```
# K-means with iris data
dat <- iris
dat$Species <- NULL # Remove Species column
km <- kmeans(dat, centers = 3)
clus <- km$cluster # Cluster assignments
dists <- dist(dat) # Distance matrix
                   # (Euclidean distance between
                   #  each observation pair)
library(cluster)
plot( silhouette(clus, dists))
```



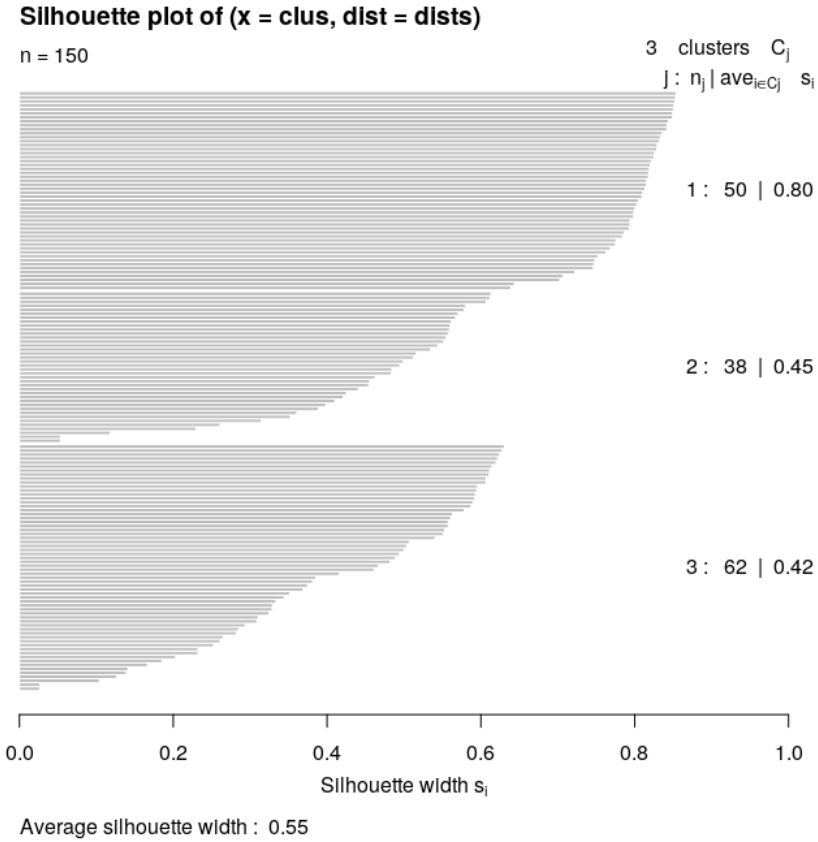
- For each of the three clusters we get the average silhouette width (0.8, 0.45, 0.42)



Silhouette plot

- For each of the three clusters we get the average silhouette width (0.8, 0.45, 0.42)
- What is a good average silhouette value? Rule of thumb:

0.7 – 1.0 : Good structure found
0.5 – 0.7 : Reasonable structure found
0.25 – 0.5 : Weak structure, requires confirmation
-1 – 0.25 : Bad!

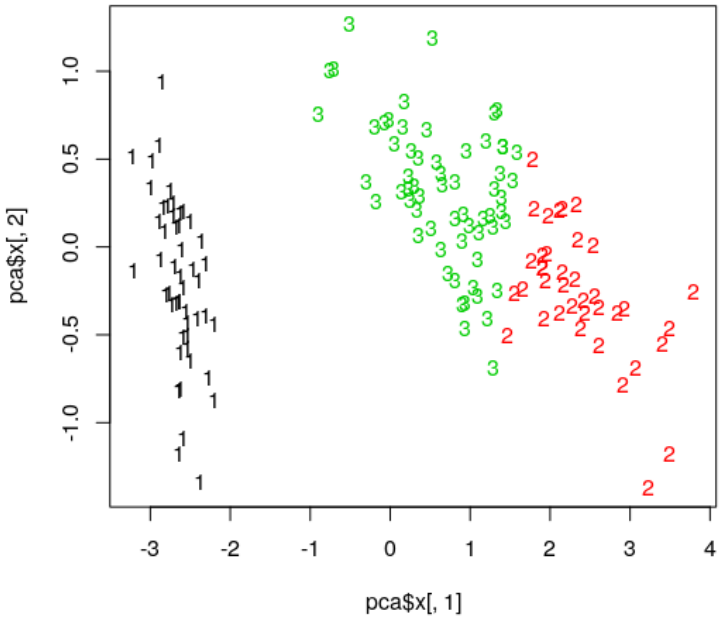


- In the example with the iris data the first cluster seems to be a clear structure. The other two clusters score less good.
- To get another impression, we could plot the **first two principal components** and look at the assigned clusters

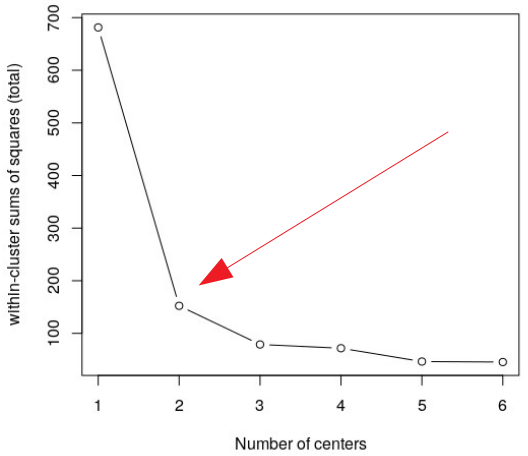
Assessing the quality of the clustering

- 2D-plot using PCA (used data is 4-dimensional)

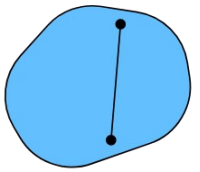
```
# PCA plot
pca <- prcomp(dat)
plot(x = pca$x[,1], y = pca$x[,2],
     col=clus, pch=as.character(clus))
```



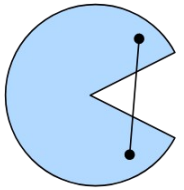
- We see the same pattern like in the silhouette plot (cluster 1 clear, the other two not so much)
- Perhaps two clusters are sufficient for the data…
  - Check within-cluster sums of squares vs. k



Problems with k-means

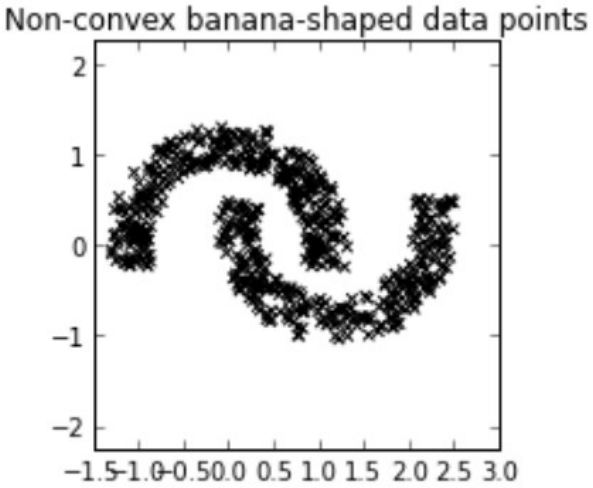


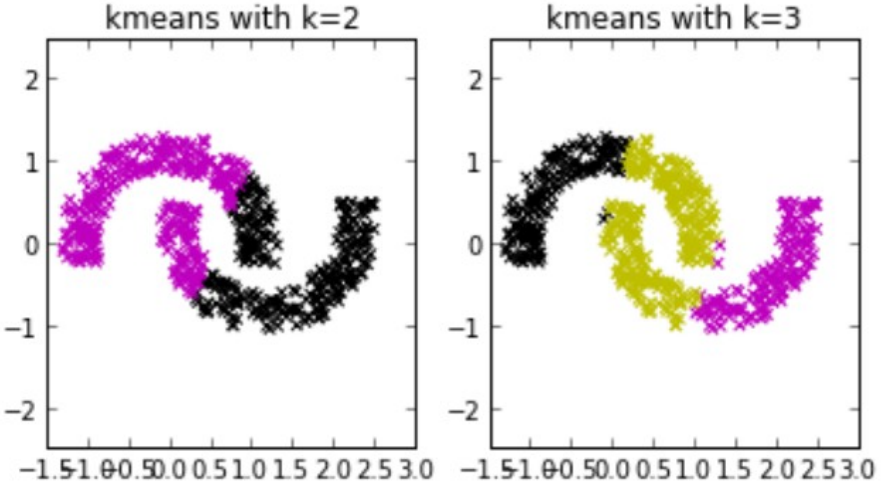
convex



non-convex

- The first step in the k-means algorithm is determined randomly
  - Results may vary due to starting configuration!
- K-means has problems clustering **non-convex data**:
  - In Euclidean space, an object is convex if for every pair of points within the object, every point on the straight line segment that joins them is also within the object (wikipedia).

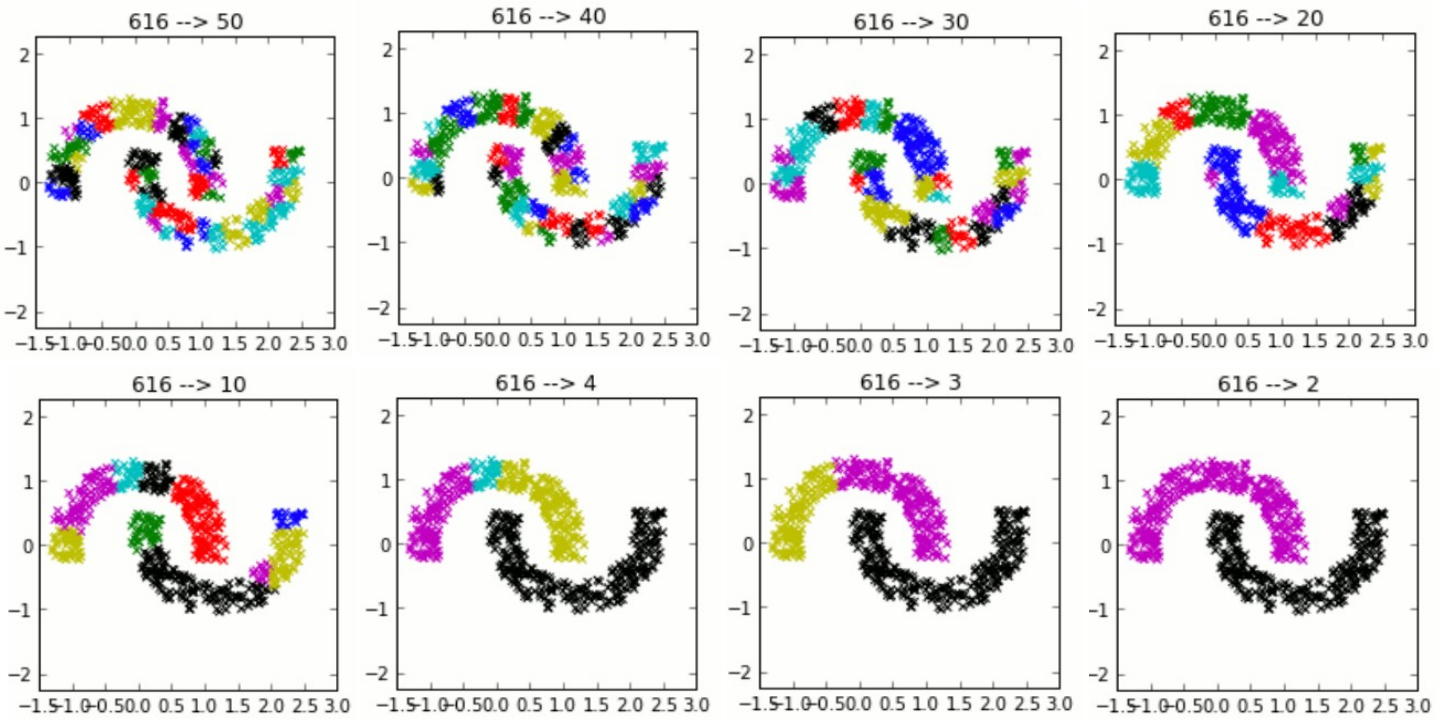


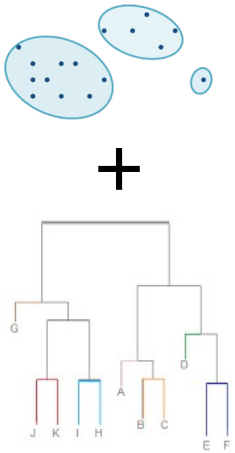


https://pafnuty.wordpress.com/2013/08/14/non-convex-sets-with-k-means-and-hierarchical-clustering/

Combining k-means with hierarchical clustering

- Issues with non-convex data can be solved by combining k-means with **hierarchical** clustering
- **Idea**: Apply k-means to data using a large **k**, then start combining small clusters into larger clusters using the **single-link** agglomerative method.
  - "Combine small clusters, which lie closest to each other"





https://pafnuty.wordpress.com/2013/08/14/non-convex-sets-with-k-means-and-hierarchical-clustering/

Summary k-means

- K-means algorithm is very **fast**
- We need to specify the number of clusters (**k**)
- Can have problems with **outliers**, and non-convex shapes
- Can converge on local optimum
- Exclusively based on Euclidean distances
- Sensitive to starting configuration