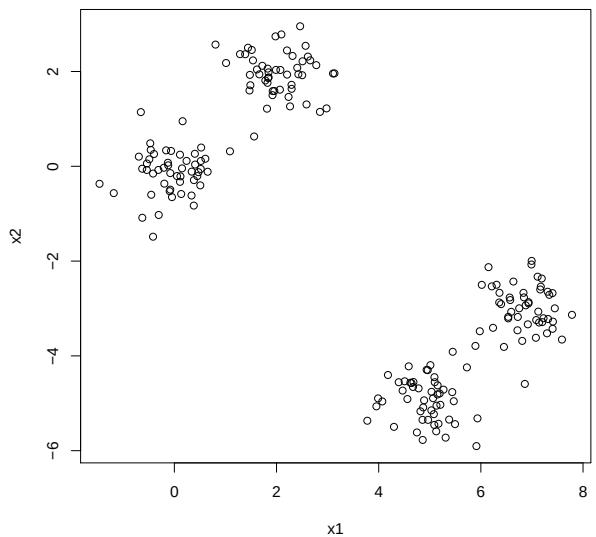
MSA220 - Statistical Learning for Big Data

Lecture 2

Rebecka Jörnsten

Mathematical Sciences
University of Gothenburg and Chalmers University of Technology

Explorative analysis - finding groups in data.
This is a more difficult task than classification since the goal is rather subjective - what is group?



Are there 2 or 4 clusters?

What defines a group is up to you to choose, e.g. by defining an object-object similarity measure or distance.
The most commonly used distance measure is euclidean distance.
However, other distances may be more appropriate to use for some data sets, e.g. matching-metrics for categorical data or correlation-based similarities for curve data or when relative differences between features are more interesting than absolute levels.

Clustering

From A.K. Jain's review paper posted on the course webpage

- What is a cluster?
- What features should be used? and how (standardized etc)?
- Outliers?
- How define similarity or distance?
- How many clusters?
- Which method? (objective)
- Validity? Are there clusters in the data?

Clustering - 4 "flavours"

- Partitioning
- Agglomerative/hierarchical
- Density based ("bump-hunting", spectral/graph-based methods)
- Model-based

Goal of Clustering - partitioning

1. Minimize within-cluster distances.
   - C denotes a *partition* that puts labels $\{1, \cdots, K\}$ on objects. $C(i)$ is the label for observation i.
   - We want the within-cluster distances to be small: $W(C) = \sum_{k=1}^{K} \sum_{C(i)=C(j)=k} d_{ij}$, where d_{ij} is the distance between objects i and j.
2. Maximize between-cluster distances.
   - Maximize $B(C) = \sum_{k=1}^{K} \sum_{C(i)=k, C(j)\neq k} d_{ij}$

Turns out the total sum of distances is $W(C) + B(C)$ so the two goals are equivalent. (But if you only consider distances to some clusters instead of all, it's not the same.)

How to generate a partition

kmeans is a very popular method and has been around for a long time. It is very simple and fast.

1. Pick K observations at random to be the cluster representatives or *centroids*, $\mu_k, k = 1, \cdots, K$.
2. Allocate observations i to the cluster whose centroid it is closest to
$$C(i) = \arg\min_k d(x_i, \mu_k),$$
where $d(x_i, \mu_k)$ is the distance between observation location x_i and centroid μ_k.
3. Update the centroids as
$$\mu_k = \sum_{C(i)=k} x_i / N_k, \quad N_k = \sum_{i=1}^{n} 1\{C(i) = k\}$$
4. Iterate until convergence (usually very fast).

Note, you may have to run the algorithm a couple of times to ensure you have converged to a local optimum due to poor choice of initial centroids.

Apart from sensitivity to starting values, kmeans is also sensitive to noise and outliers in the data (because you are computing means and use euclidean distance).
In addition, kmeans doesn't try to take cluster shape into account and tends to find spherical groups (cf. nearest-centroid classifier).
How many clusters? You can track how much the within-cluster distances decrease as a function of the number of clusters. Once you start adding more clusters that the data "supports" there is very little decrease in $W(C)$ since you are forcing kmeans to split close observations into smaller groups. You should look for a point where the $W(C)$ levels off as the number of clusters grow (see demo code).

K-MEDOIDS, PAM

PAM, partition around medoids, is an alternative to kmeans that is more robust and allows for extensions to non-euclidean distances. Instead of using centroids (cluster means) to summarize a group, we use an observation = a medoid. Medoid is a multivariate generalization of a median.

1. Pick K observations at random to be the starting medoids, $m_k, k = 1, \cdots, K$
2. Allocate observations to the cluster with the closest medoid.

$$C(i) = \arg\min_k d(x_i, m_k)$$

3. Update the medoid as the observation in the cluster with the closest distance overall to all other observations in the cluster

$$m_k = \arg \min_{i:C(i)=k} \sum_{C(j)=k} d_{ij}$$

4. Run until convergence

K-MEDOIDS, PAM

PAM, partition around medoids, is an alternative to kmeans that is more robust and allows for extensions to non-euclidean distances. Instead of using centroids (cluster means) to summarize a group, we use an observation = a medoid. Medoid is a multivariate generalization of a median.

1. Pick K observations at random to be the starting medoids, $m_k, k = 1, \cdots, K$
2. Allocate observations to the cluster with the closest medoid.

$$C(i) = \arg\min_k d(x_i, m_k)$$

3. Update the medoid as the observation in the cluster with the closest distance overall to all other observations in the cluster

$$m_k = \arg \min_{i:C(i)=k} \sum_{C(j)=k} d_{ij}$$

4. Run until convergence

Some things to note about PAM

- The entire algorithm uses only object-object distances d_{ij}
- No new distances need to be computed (to centroids etc since medoids are already part of the set of observations).
- Input to PAM can thus be any kind of pairwise distance matrix! Lots of choices possible.

Silhoutte Width

The authors of PAM also suggested a new way to select the number of clusters for the data set. For each observation i compute

- $a_i = \sum_{j:C(j)=C(i)} d_{ij} / sum_{j:C(j)=C(i)} 1$, the average distance to all members of the same cluster as observation i.
- $b_i = \min_{k' \neq C(i)} \sum_{j:C(j)=k'} d_{ij} / \sum_{j:C(j)=k'} 1$, the average distance to members of the nearest cluster to the cluster i belongs to.
- $s_i = \frac{b_i - a_i}{max(b_i, a_i)}$ is the silhouette for observation i and is a value between -1 and 1.

The closer to 1 s_i is, the better observation i clusters within its own cluster. The smaller s_i is, the closer i is to being clustered with another group. A negative s_i indicates that i is on average closer to members of another cluster than the one it's been allocated to which can happen if i is near the cluster boundary and one cluster is more spread out than the other. Small or negative values for s_i are warning signals that the clustering may be poor.

Silhoutte Width

For a good clustering results we want all observations to be well-clustered. We thus focus on the average silhouette value, $S = \sum_i s_i / N$.
Pick the number of clusters K that maximizes the silhouette.
Drawbacks? Can only pick $K > 1$ clusters. The silhouette tends to be a bit conservative (picking few clusters) and struggles in settings where clusters are of very different scale (some very spread out and some very densely clustered).
Also, like kmeans, PAM and silhouette width look for spherical clusters.
PAM is implemented in the cluster package in R which also contains many other algorithms. See the paper by Struyf et al or the cluster package documentation.

Cluster prediction strength

Another method for choosing the number of clusters in a data set draws upon classification methods.
The idea is as follows: A clustering with the "correct" number of clusters is something that should be based on non-random structures in the data. Therefore the finding of groups should be reproducible - similar groups should be found if you were able to obtain a new, independent draw of data from the same data generative distribution.

Cluster prediction strength

For a given number of clusters, K

1. Divide the set of observations into two parts: A and B
2. On each of the data sets cluster the observations into K groups. Call these partitions C_A and C_B respectively
3. The partitions results in a labeling of the data sets. Treat these labels as "true" labels and learn a classification rule for each of the data sets: rule c_A is learnt from data set A with class labels C_A, rule c_B is learnt from data set B with class labels C_B.
4. Use data set B as a test set for the classifier c_A and data set A as the test set for classifier c_B. That is, the rule c_A applied to data set B results in a new labels $c_A(B)$ to be compared to the cluster label C_B and v.v. for data set B. Note, you may need to do a label-matching/permuting of labels first. Since the order of labels is arbitrary, group 1 in data set A might correspond to group 4 in data set B etc.
5. Compute the overall test error rate as the average of the test error rate in data set A and data set B

Cluster prediction strength

The optimal number of clusters is the K that makes the classifier from each data set predict the cluster label on the other data set as best as possible (reproducible groups).
If you try to find more clusters than is supported by data, the clusters are not reproducible since the "extra" clusters will correspond to some kind of random division of a group which will not line up for the different data sets.

Cluster prediction strength

Some things to consider:

- You need a lot of observations for this to work - enough such that cluster structure can be seen with only 50% of the data.
- Think about which clustering method and which classifier goes together. Kmeans and nearest-centroids are a good match. kmeans and LDA is also an OK match. If you use cluster methods based on non-euclidean distance remember you have to have a classifier that works like that also (kNN for example).

Hierarchical clustering

Hierarchical clustering is very popular since it is simple, intuitive and comes with a nice graphical display. Like PAM, it takes pairwise distances as input which makes it rather flexible.
In contrast to PAM and kmeans it constructs clusters "bottom-up", i.e. building clusters by joining observations together as opposed to splitting the data into groups ("top-down").

Hierarchical clustering

1. Start with all the observations as their own clusters, $g_1, g_2, \cdots, g_n$, each cluster of size 1.
2. Join the pair of clusters g_i and g_j that are the closest together
3. Keep on joining clusters pairs until all observations are in one big clusters of size n.

Step 2 involves some subjective choices:
what is close? that is, what kind of distance metric do you want to use?
what is meant by clusters being close? that is, how do we combine information about observation pairwise distances into a group-level distance?

Linkage

Cluster-cluster distance is called *linkage*

- **average linkage** is the most commonly used. The distance between clusters g and h is computed as

$$d_{gh} = \sum_{i:C(i)=g, j:C(j)=h} d_{ij} / \sum_{i:C(i)=g, j:C(j)=h} 1$$

  The average similarity between all pairs in the two different clusters is encouraged.

- **Single linkage** is not used very often since it tends to create clusters that are quite spread out.

$$d_{gh} = \min_{i:C(i)=g, j:C(j)=h} d_{ij}$$

Two clusters can join as long as there is a pair of observations, one from each cluster, that is close.

- **Complete linkage** is popular since it tends to produce very "tight" clusters

$$d_{gh} = \max_{i:C(i)=g, j:C(j)=h} d_{ij}$$

Two clusters can only join if the "worst pair" of observations is close enough.

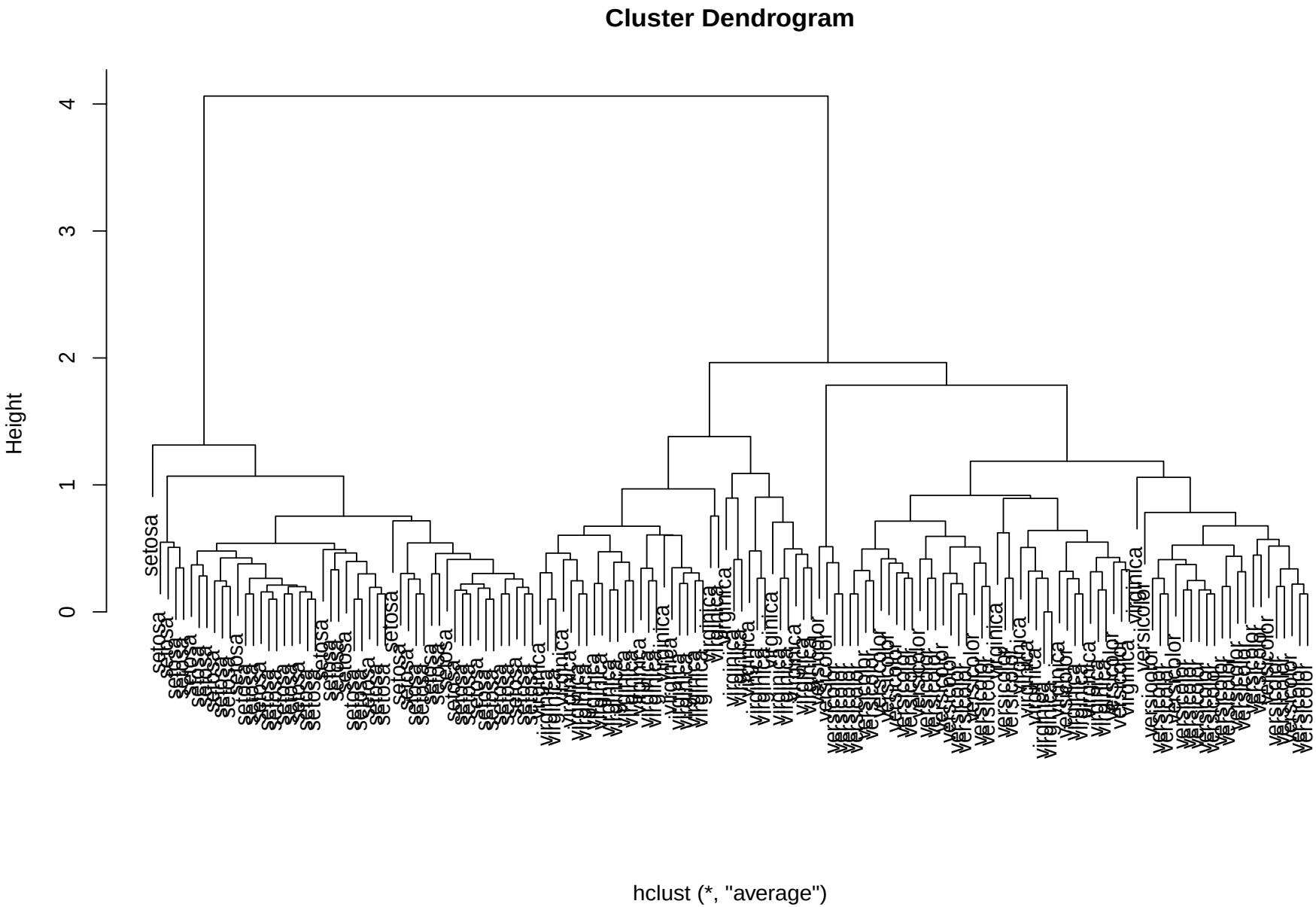
Linkage

Average and complete linkage are the most popular BUT which linkage is the most suitable depends on the data set at hand and what you want clustering to achieve (remember the iris data demo from class).
It is always a good idea to run with different linkages and compare the results.

Dendrogram

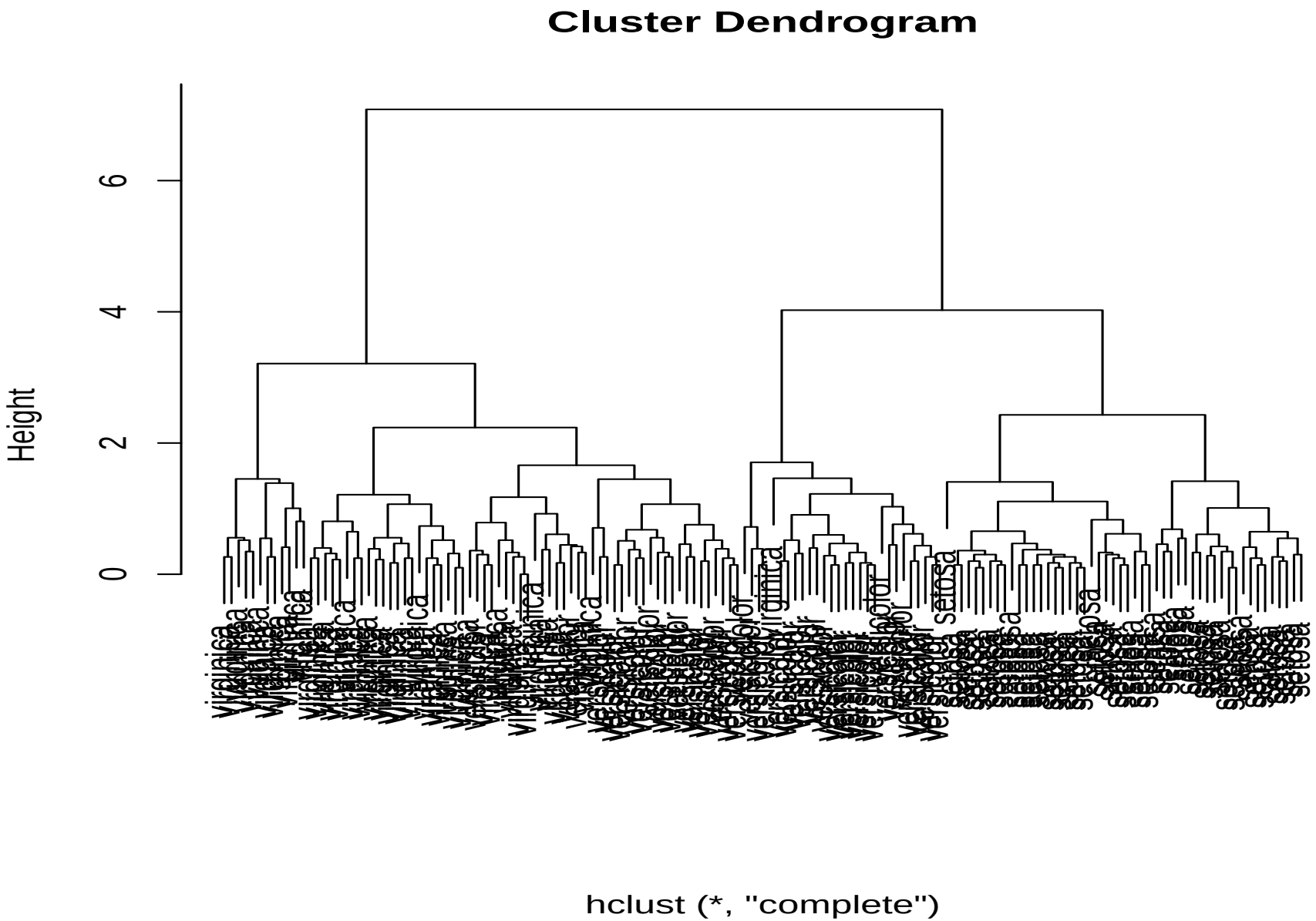
Hierarchical clustering is graphically summarized with the *dendrogram*. This depicts the iterative procedure of joining clusters.
The dendrogram looks a bit like a CART tree but the meaning is different. You read the dendrogram from the bottom-up, this is how the clusters are formed. The length of the branches in the dendrogram represents the cluster-cluster distances. A really long branch indicates that the within-cluster distances were increased a lot by joining the cluster at the bottom of the branch with the other cluster at the top of the branch.
The dendrogram can therefore suggest how many clusters you should form from your data. Long branches can be cut to form distinct group that have small within-cluster distance and is well separated from the rest of the observations.

Hierarchical clustering



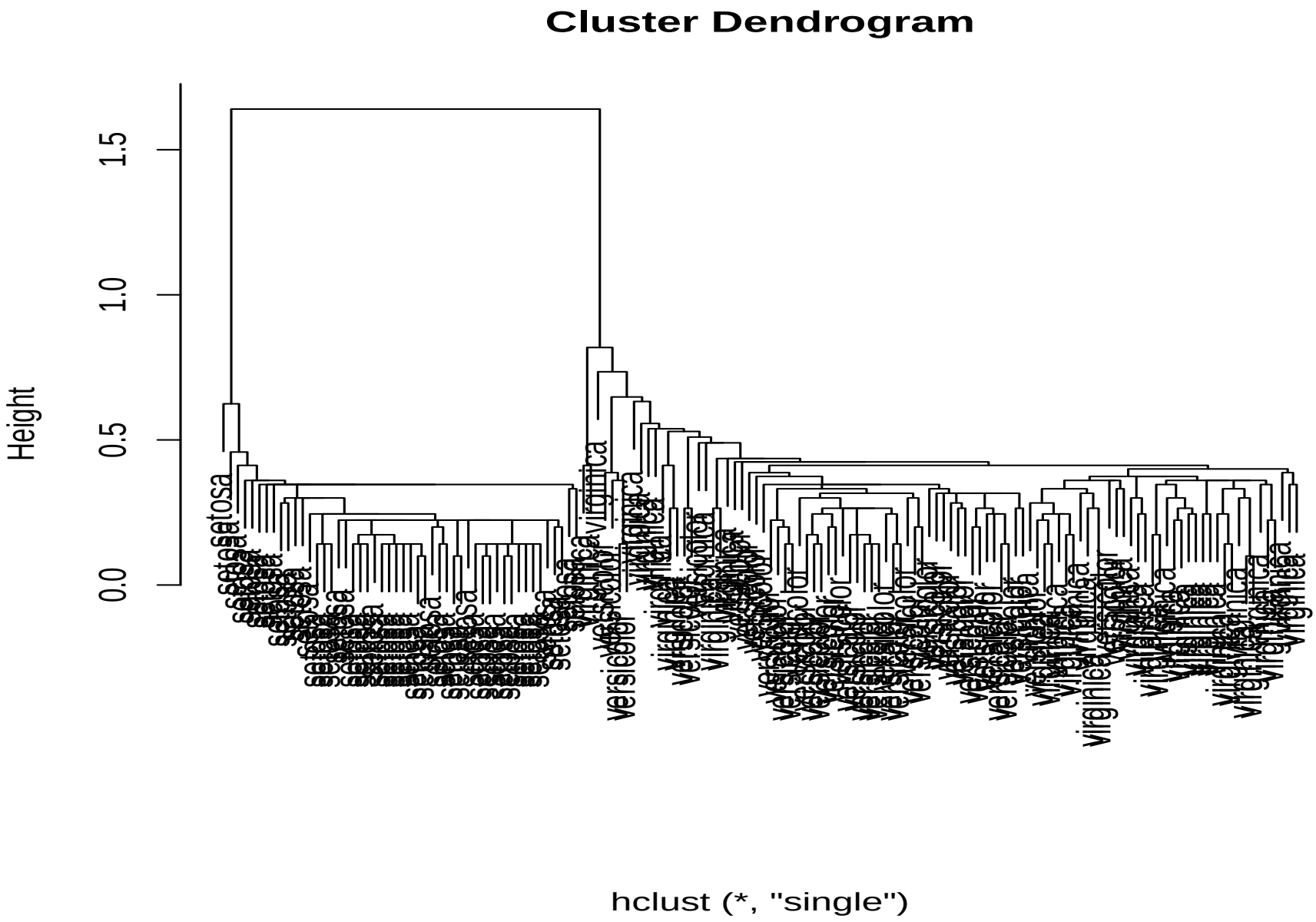
The average linkage identifies two groups of irises (long branches in the dendrogram): setosa and the versicolor/virginica. The latter group is very mixed up.

Hierarchical clustering



The complete linkage identifies mainly two groups also, but here setosa is mixed up with some other virginica and versicolor. If you cut the dendrogram into three groups setosa is separated out.

Hierarchical clustering



Single linkage identifies two clusters: setosa and virginica/versicolor. You can see in the dendrogram how the clusters are built from the joining of observations to an already formed cluster rather than clusters joining clusters.

Modelbased clustering

So far we have looked at nonparametric cluster methods - clusters are defined through a distance metric and a construction algorithm/criterion. We have noted that clustering is a difficult problem because these choices are subjective.
Parametric, or modelbased clustering, takes clustering into a familiar statistical modeling framework where we can say something about the goodness-of-fit of clusters. It is not a statistical problem that can be objectively analyzed BUT of course relies on a modeling assumption that is a subjective choice nonetheless.

So on the plus-side, we make clear modeling assumption and can validate them and analyze/draw inference from results.
On the minus-side, our modeling assumption can be too strict, not flexible enough to capture groupings in the data.
The most commonly used modeling assumption in modelbased clustering is that each group is multivariate normally distributed, that is, groups are ellipsoid blobs in x-space.

EM-iterations

- If we knew the labels, we could estimate the parameters of each cluster easily: just compute the mean and the covariance for each group.
- If we knew the model parameters, we could allocate observations to each cluster using the posterior probability approach: allocate to the model that best describes the object (likelihood and perhaps a prior).

This iterative process is the EM approach to model fitting and is a method used to solve complex estimation problems that would be easy to solve with some additional information (as done in each step or the iterative procedure).

We have

$$x_i \sim \sum_{k=1}^{K} \pi_k \Phi(x_i \mid \mu_k, \Sigma_k)$$

where Φ is the multivariate-normal density, π_k is the unknown proportion of each cluster in the data, μ_k is the cluster center and Σ_k the spread of the observations in each cluster around the cluster center.

$$\Phi(x_i \mid \mu_k, \Sigma_k) = \frac{1}{\sqrt{2\pi \mid \Sigma_k \mid}} exp(-\frac{1}{2}(x_i - \mu_k)' \Sigma_k^{-1} (x_i - \mu_k)$$

Let's try to estimate the parameters via maximum likelihood.
Likelihood:

$$L(x_1^n \mid \pi, \mu, \Sigma) = \prod_{i=1}^{n} \left(\sum_{k=1}^{K} \pi_k \Phi(x_i \mid \mu_k, \Sigma_k) \right)$$

Log-likelihood

$$l(x_1^n \mid \pi, \mu, \Sigma) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \Phi(x_i \mid \mu_k, \Sigma_k) \right)$$

D'Oh! a sum inside the logs - this makes things difficult to solve in closed form.

Informal derivation of EM:
Try taking derivatives of the log-likelihood function by application of the chain-rule.
For example, focus on the k-th cluster mean, μ_k.
We have

$$l(\mu_k) = f(g(h(\mu_k)))$$

where $f()$ is log(), $g()$ is $\sum_{l=1}^{K} \pi_l \Phi_l()$ and $h()$ is $\Phi_k()$. Then the chain-rule gives us

$$\frac{\partial l}{\partial \mu_k} = \sum_{i=1}^{n} \frac{1}{\sum_{l=1}^{K} \pi_l \Phi(x_i \mid \mu_l, \Sigma_l)} * \pi_k \Phi(x_i \mid \mu_k, \Sigma_k) * \Sigma_k^{-1}(x_i - \mu_k)$$

where the first term is the $\frac{\partial f}{\partial g}$, the second $\frac{\partial g}{\partial h}$ and the third $\frac{\partial h}{\partial \mu_k}$.

We set this derivative to 0:

$$\sum_{i=1}^{n} \eta_{ik} \Sigma_k^{-1} (x_i - \mu_k) = 0$$

where

$$\eta_{ik} = \frac{\pi_k \Phi(x_i \mid \mu_k, \Sigma_k)}{\sum_{l=1}^{K} \pi_l \Phi(x_i \mid \mu_l, \Sigma_l)}$$

Solving for μ_k we get

$$\hat{\mu}_k = \frac{\sum_{i=1}^{n} \eta_{ik} x_i}{\sum_{i=1}^{n} \eta_{ik}}$$

Problem? η_{ik} actually depends on μ_k so we can't solve for μ_k like this. However, if we have an estimate of η_{ik} based on previous estimates of μ_k this could work.
This is exactly what the EM does. $\eta_{ik} = P(C(i) = k \mid x_i, \pi, \mu, \Sigma)$ is the posterior probability that observation i belongs to cluster k given the data and all the model parameters.
We will start with an initial guess for the η_{ik}, e.g. a k-means clustering of the data, and then estimate the model parameters, update the η and iterate. Convergence is usually pretty fast, depending on the starting cluster allocation, dimensionality of the data and noise level.

The M-step estimate the model parameters given the posterior probability estimates η:

$$\hat{\mu}_k = \frac{\sum_{i=1}^{n} \eta_{ik} x_i}{\sum_{i=1}^{n} \eta_{ik}}$$

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^{n} \eta_{ik} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)'}{\sum_{i=1}^{n} \eta_{ik}}$$

$$\hat{\pi}_k = \frac{\sum_{i=1}^{n} \eta_{ik}}{\sum_{i=1}^{n} 1}$$

The E-step updates the posterior probabilities as

$$\eta_{ik} = \frac{\pi_k \Phi(x_i \mid \mu_k, \Sigma_k)}{\sum_{l=1}^{K} \pi_l \Phi(x_i \mid \mu_l, \Sigma_l)}$$

Cautionary remarks:

- EM is sensitive to the choice of starting values and can converge to a local optimum or exhibit very slow convergence if starting values are poorly chosen. Track the likelihood as a function of the iterations to catch this and try a couple of different starting values.
- EM applied to MVN mixture modeling has a tendency to create empty clusters or singleton clusters which creates singular Σ_k. Once a cluster starts growing, Σ_k often increases in the next M-step which makes it even easier for an observation to be allocated to cluster k in the next E-step, etc.

We deal with the singularities by regularizing the estimates of the Σ_k. That is, we estimate

$$\tilde{\Sigma}_k = \hat{\Sigma}_k + \lambda\Lambda$$

where Λ is a covariance matrix you regularize Σ_k toward, usually taken as a scaled version of the global covariance (covariance of data without clustering) and λ controls how much you regularize. This form of regularization has a Bayesian motivation, and is the Bayesian covariance estimate if you make a prior assumption on $\Sigma_k = \Lambda$ with an Inverse-Wishart prior distribution.
Don't regularize too much - keep λ as small as possible so that the data dominates the likelihood and not the prior.

The EM-algorithm outputs parameter estimates and posterior probabilities η_{ik}. A final cluster allocation is achieved as

$$C(i) \arg\max_k \eta_{ik}$$

but you can also use the η directly as your "soft-clustering" output. How many clusters should you use?

As in all model-based methods, the likelihood cannot be used to select the model (number of clusters) as the likelihood is always increased by adding more and more model parameters to the description of the data - i.e. using the likelihood to select the number of clusters would just lead you to choose the largest number of clusters you try.
However, as we are in a standard modeling setting we can use the off-the-shelf model selection criteria that you may be familiar with from regression. The most commonly used criterion in mixture modeling is the Bayesian information criterion, BIC.

$$BIC(K) = -2 * \log - likelihood + \log(n) * p$$

where

$$p = (K - 1) + K * D + K * D(D + 1)/2$$

is the number of model parameters, where the three terms refer to the number of π's, μ's and Σ's respectively.
The BIC measures the trade-off between the model fit to the data (the loglikelihood term) and the complexity of the model (the number of parameters).

You pick the number of clusters K that minimizes the BIC.
In practice, BIC is usually rather flat-looking so there are often a couple of different values for K that gives you an almost equally good fit.
The number of parameters in your model can be altered if you are willing to make simplifying assumption on the cluster shapes, sizes or correlation structures (i.e. on the form of Σ_k). If you simplify the cluster shapes you save on the number of parameters and may "afford" more clusters.

This idea of simplifying the cluster shapes, sizes and correlations (orientations) was the basis for the Mclust procedure of Raftery et al (2006). This modelbased clustering method has been implemented in a very easy-to-use R-package (`mclust()`).

Setup:

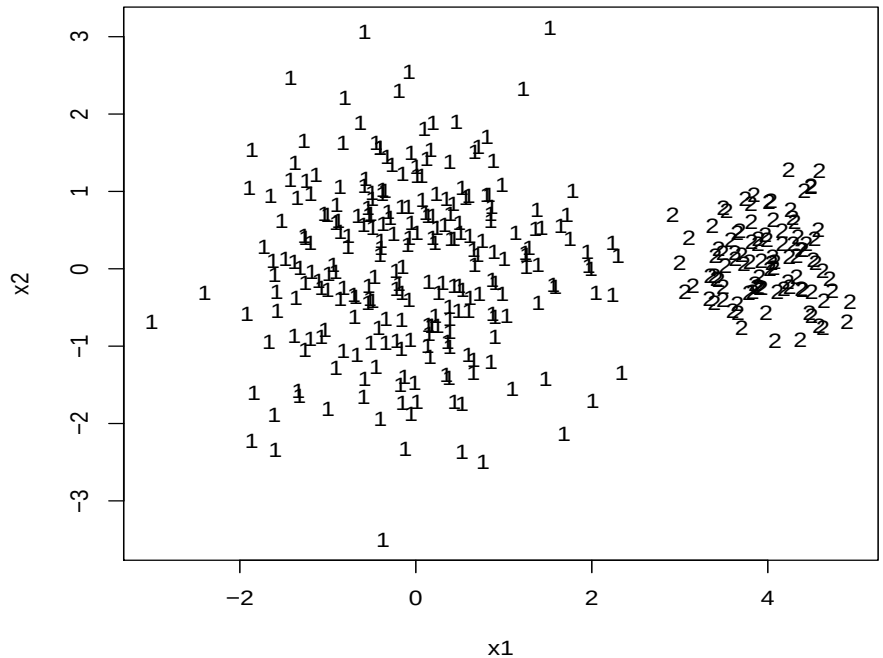
$$x_i \sim \sum_{k=1}^{K} \pi_k \Phi(x_i \mid \mu_k, \Sigma_k)$$

where we write

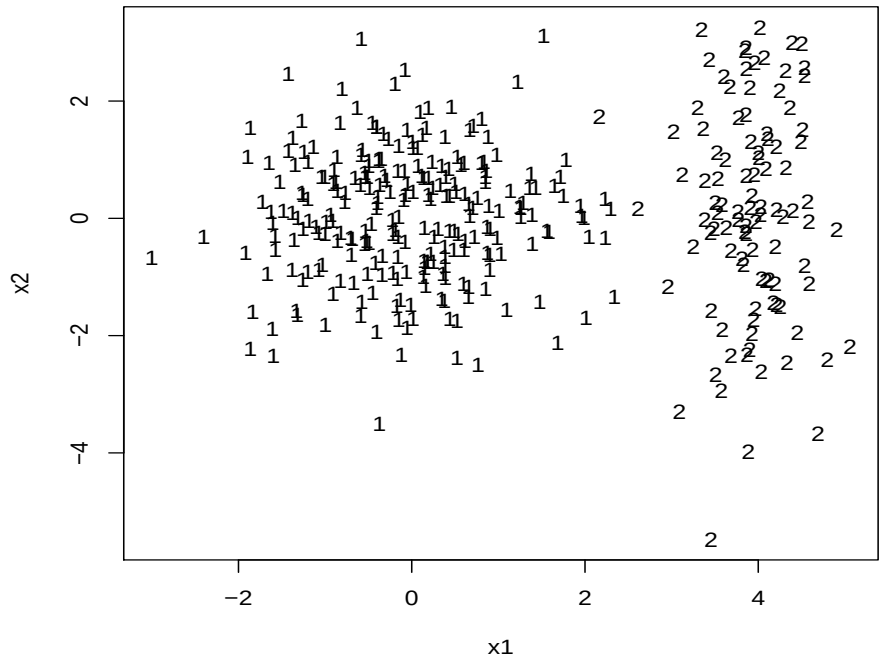
$$\Sigma_k = \lambda_k D_k A_k D_k'$$

This is the eigenvalue decomposition of Σ_k.

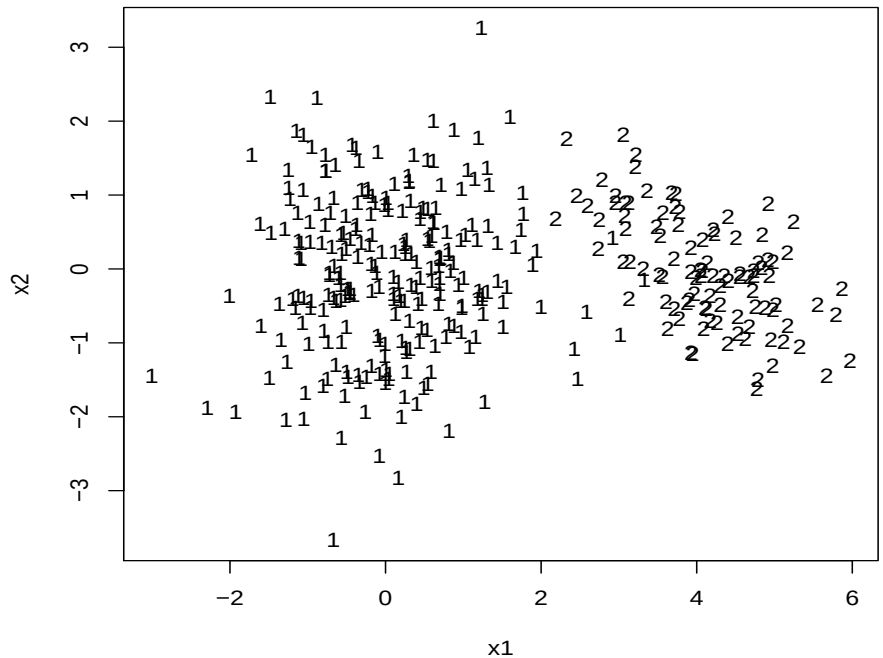
- λ_k is a scalar that controls the *volume* of the cluster
- D_k is a matrix that controls the *orientation* (correlation structure) of the cluster
- A_k is a diagonal matrix that controls the *shape* of the cluster, i.e. the relative spread of each feature.



$\lambda_1 > \lambda_2$ so the first cluster has a bigger volume than the second.



For cluster 1: $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ so both features have equal spread, and for cluster 2: $A = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$, i.e. much more spread in feature 2 than feature 1.



For cluster 1: both D and $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ so both features have equal spread and no correlation, and for cluster 2: $A = \begin{pmatrix} 1 & 0 \\ 0 & .25 \end{pmatrix}$, $D = \begin{pmatrix} 0.7 & -0.7 \\ 0.7 & 0.7 \end{pmatrix}$ i.e. $\Sigma \simeq \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$, i.e. the features are negatively correlated in cluster 2.

In Mclust you can set some or all of the λ's, A's and D's to be equal across clusters. This can save a lot of parameters but of course also corresponds to making assumptions about the clustering structure in the data (e.g. that feature dependencies is the same for all clusters, $D_k = D$).
Some examples:

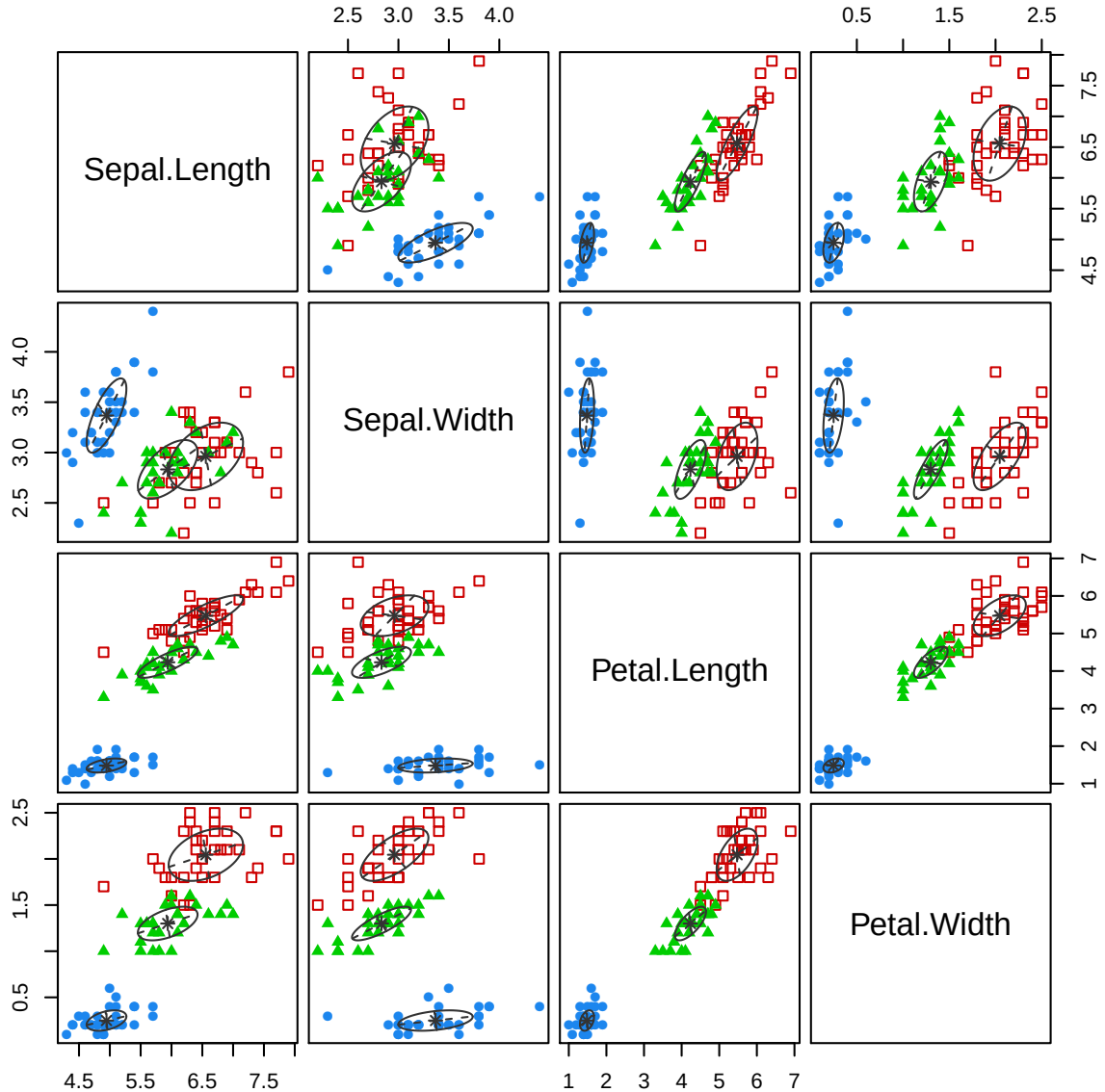
- $\Sigma_k = \Sigma = \lambda I$ assumes that clusters are equal volume, spherical blobs in x-space
- $\Sigma_k = \lambda_k I$ assumes that clusters are different volume but spherical for all clusters
- $\Sigma_k = \lambda A$ assumes that all clusters have the same volume, spread can vary for different features but in the same way for all clusters, and there are not feature-feature dependencies

Some more examples:

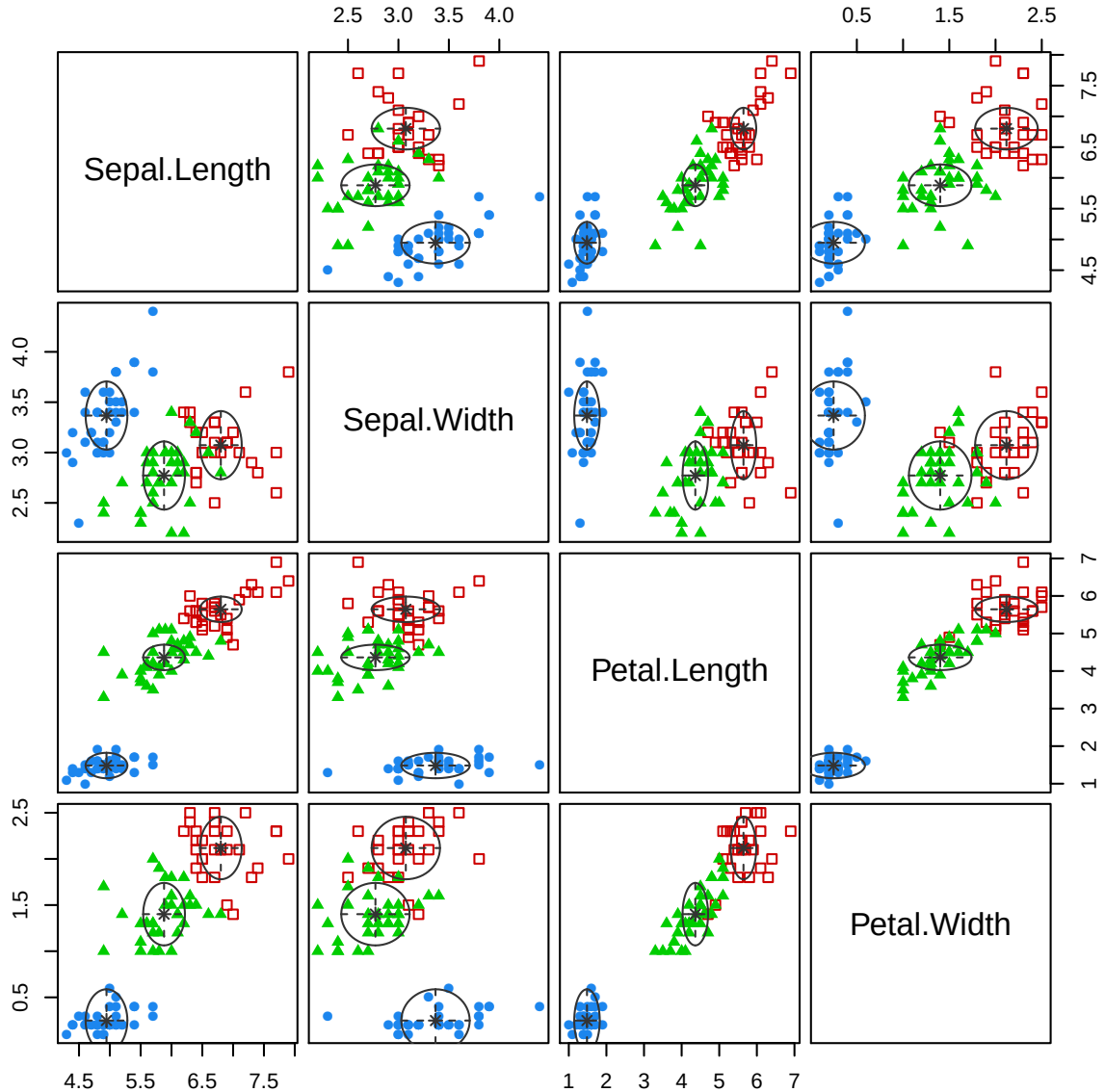
- $\Sigma_k = \lambda A_k$ assumes all clusters have the same volume but can have different spread for different features (e.g. feature 2 more variable than feature 1 in cluster 1, and the opposite is true for cluster 2.)
- $\Sigma_k = \lambda DAD'$ assumes that the volume, shape (feature spread) and orientation (feature dependencies) are the same for all clusters

We can use BIC to select between these special cases for the clusters since we simply count the number of parameters in each model and add to the goodness-of-fit (negative log-likelihood).

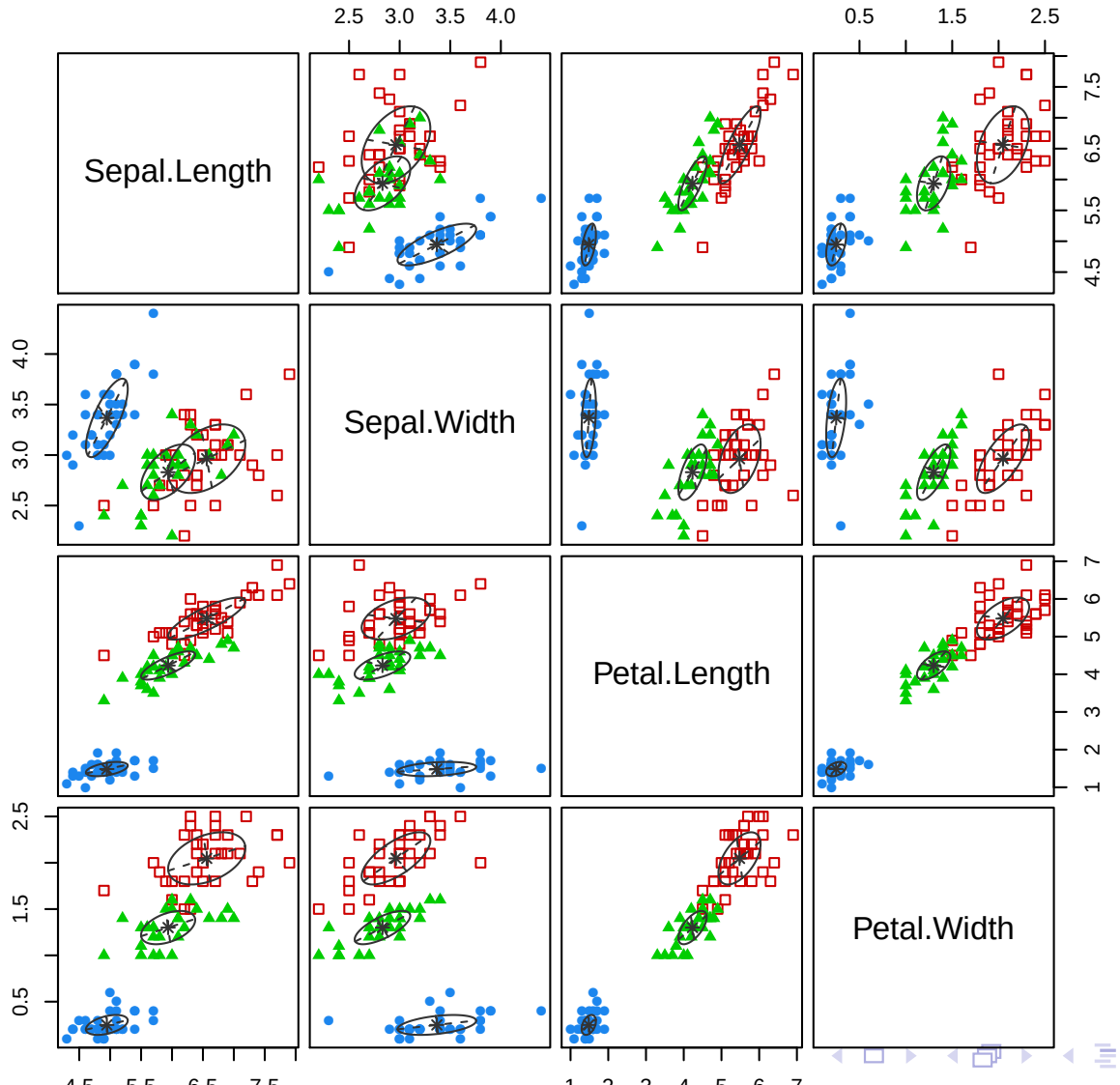
Mclust with varying volume, shape and orientation



Mclust with equal volume, spherical distributions



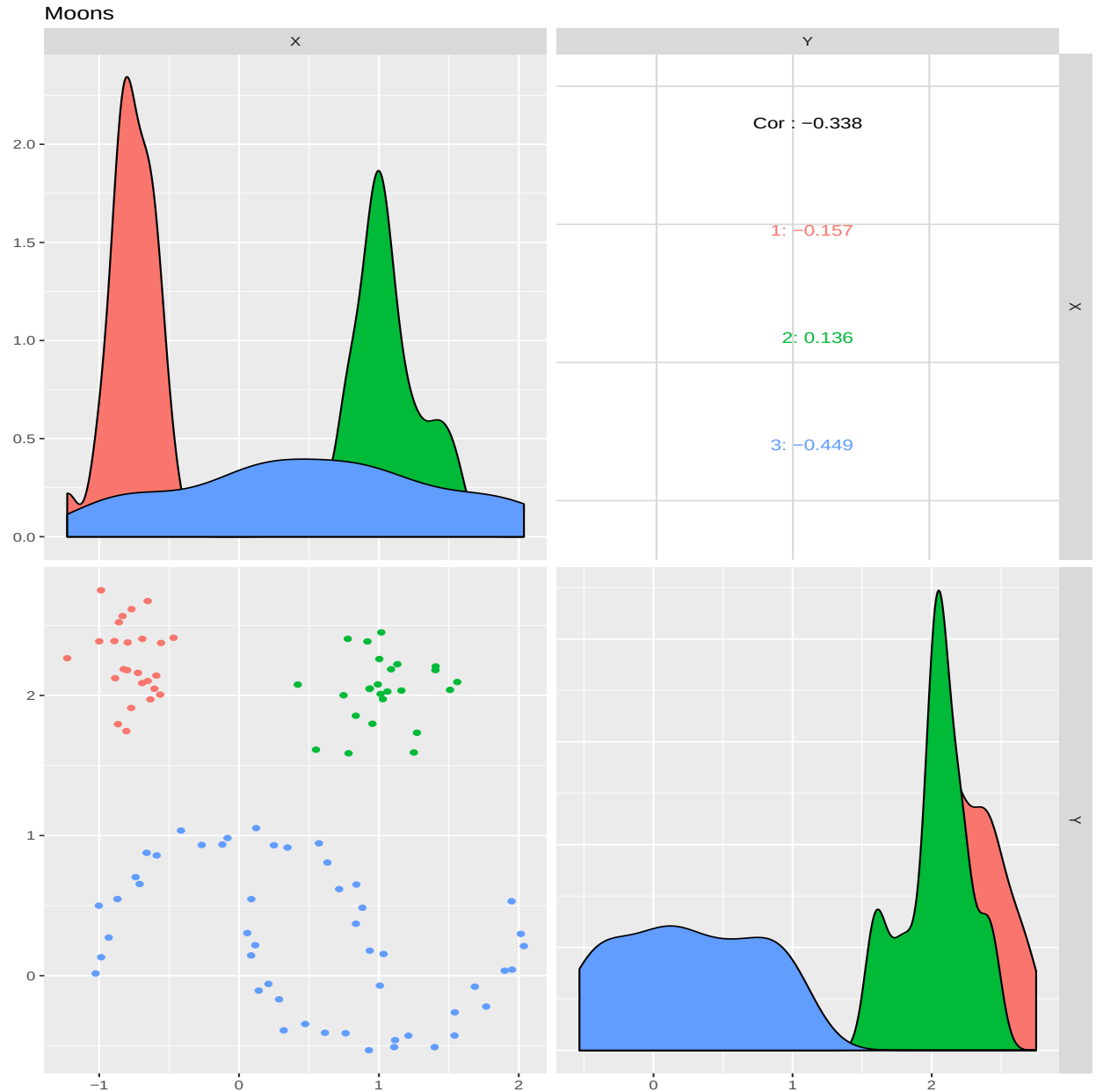
BIC selects varying volume, equal shape and varying orientation for the clusters.



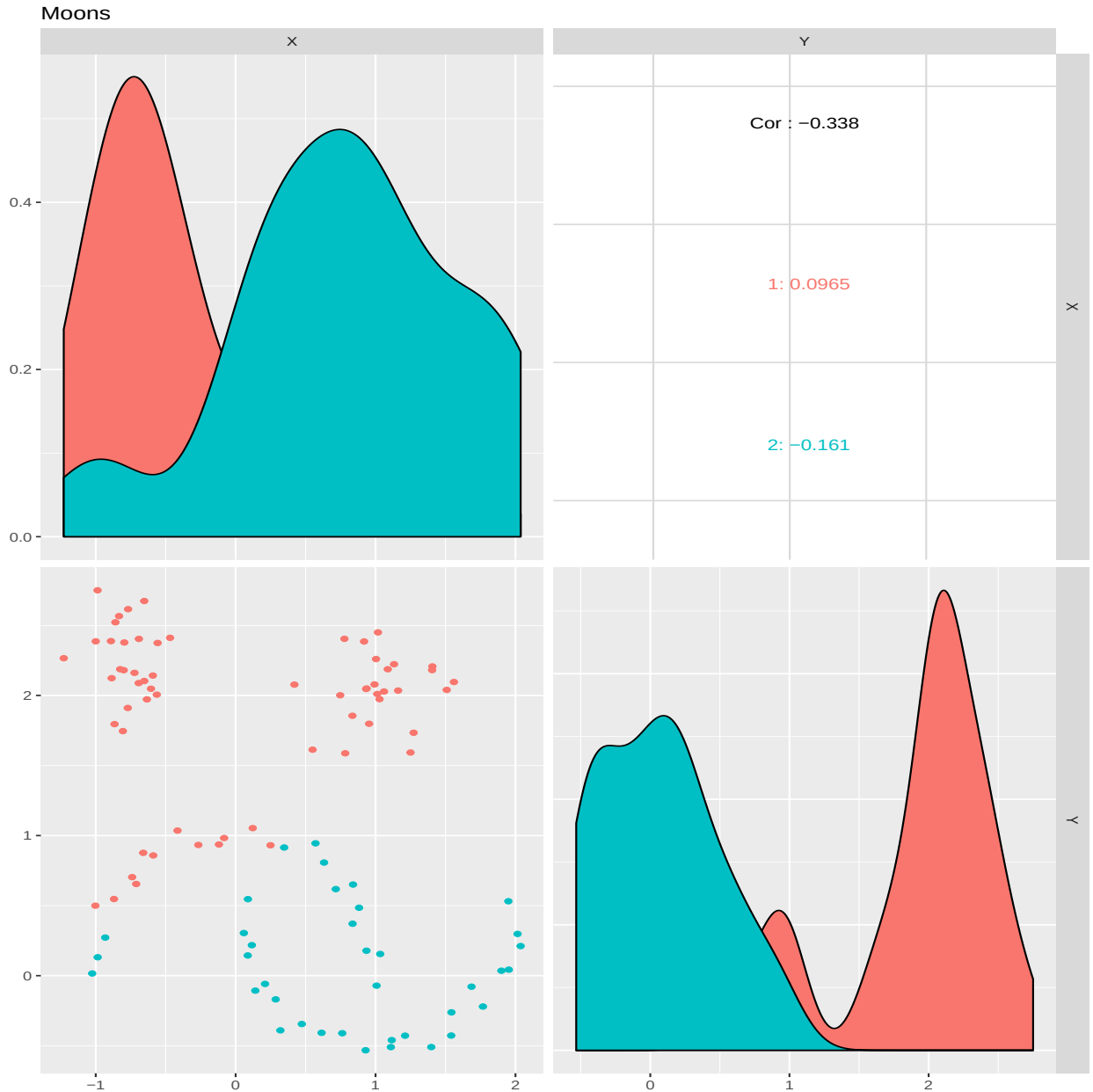
Density-based clustering

- It can be difficult to formulate a distribution that can handle "odd shapes" of groups
- Distance-based methods are sometimes "too global"
- Use only distance information between observations that are close instead.

Density-based clustering



KMEANS



- Ester et al, 1996
- Using a chosen distance metric, identify so-called *core points* that have within an ϵ-distance a minimum of *minP* points.
- Find the connected components of core points, i.e. core points that can be reached from one another via other core points.
- For non-core points, assign them to a component from which it can be reached (i.e. within ϵ distance from a core point

(The last step can lead to a non-unique assignment)

- ϵ controls how much of the data will be clustered. If you pick this too small you create isolated islands in the data, too big and everyone is connected. You can think of ϵ as a max distance before you don't consider observations to be similar or close anymore.
- *minP* is how you control how easy or difficult it will be to connect components. If you pick this as 2 you revert pack to a nearest neighbor setting which is a special case of hierarchical clustering, too big and you obtain very few core points from which to build clusters and again, many observations are not clustered. A cluster will, by definition, contain at least *minP* points.

- How pick *minP*?
- The authors of the original paper argue that choice of *minP* has limited impact. It's not what I've seen in practice. Tricky with high-dimensional data also. And depends from data to data.
- If you pick *minP* "too large" many observations will be labeled as noise (unclustered). A small *minP* can result in many small clusters.

- How pick ϵ?
- Logic: let's say the distance between a point and its k-th nearest neighbor is d. Then a d-neighborhood of the point contains $k + 1$ points.
- If data are clustered (cluster size $>$ k) then changing k won't change d much - a cluster defined as a dense set of observations.
- Now we look the kNN distances for all points in the data set. The role of ϵ is to sort out what is noise and what is structured. If there is a group of observations with much larger kNN distances, this indicates a break point and you can choose this as your ϵ (kNNdistplot in R).
- If the kNNdist looks "odd" - like a stepfunction - this suggests that the density is quite different for different parts of the data/clusters. Extensions to deal with this will be discussed later in the class.

- A generalization of dbscan: hierarchical dbscan (Campello et al, 2013)
- Only *minP* as a tuning parameter - a core distance is defined as the distance from a point to its *minP* nearest neighbor
- Then you can check if this distance is below any threshold ϵ in which case the point is an ϵ-core point (as defined in dbscan)
- The reachable distance between points p and q is defined as the maximum of their core distances and the distance(p,q).
- Construct a graph that connects all points with an edge-weight corresponding to the reachable distance
- Any ϵ-dbscan can now be obtained by running hierarchical clustering with single linkage on the graph-distance matrix and cutting the tree at ϵ - you can use the dendrogram to decide which ϵ makes sense for your data. Some observations may form a singleton cluster and they are labeled as unclustered.

PROJECTS!!!

You can work in pairs and pick from any of the topics below - but at most 2-3 teams per topic - it's more fun for everyone if we spread out a bit. You sign up for the project at this doodle.
We will present our findings in class on April 12th and this way you all get to learn about the other topics. Prepare a short presentation (6 or so slides).
There are tons of datasets available online. Example: "Interesting data for statistics", "Awesome public data sets" and of course sites like the UCI ML data repository

PROJECTS!!!

1. **How many clusters?**: On real or simulated data, compare some methods for choosing the number of clusters. The R package(nbclust is a good place to start.
2. **Imbalanced data**: What if there are large and small clusters intermixed? Try different clustering methods and compare performance. What is the "break-down" point for each method?

PROJECTS!!!

3. **Unrelated features**: What if your data contains clusters but these are determined by only a few features and most of the features are unrelated to the clustering?
4. **Unsupervised Random Forest**: Steve Horvath and co-authors proposed using a classification method for clustering. Check out this paper and R package. Does it work?

PROJECTS!!!

5. **Big data clustering**: Look for R packages (or python) to cluster big data sets (high-dimensional or large number of observations). Investigate the performance - pros/cons/works as advertised?
6. **Which distance? Which method?** Clustering is a function of a chosen distance and a method/algorithm that uses that distance to define groups. Compare on different data (real or simulated) how much of an impact the choice of distance metric paired with a choice of clustering method can have. Consider scales of features, types of features (binary, continuous) etc. (For mixed data the function daisy() in the cluster package uses the so-called Gower distance.)