Lab 06: Cluster Validation

PSTAT 131/231, Spring 2021

Learning Objectives

- Review `kmeans()`, `pam()` and `hclust()` and visualize clusters
- Learn package `dendextend` to produce a more readable dendrogram
- `NbClust()`
  - Perform kmeans/hierachical clustering with different distances and indices
  - Select the optimal number of clusters
- Cluster similarity comparison
  - Calculation of distance matrix (Lab 1 and Lab 5)
  - Compare by a simple table (Lab 5)
  - Silhouette plot for PAM
  - Visualize distance matrix for hierarchical clustering and k-means

---

1. Review three clustering methods of Lab05

Last time, we learnt how to perform k-means, k-medioids and hierarchical clustering by `kmeans()`, `pam()`, and `hclust()`. Let's have a quick review of these methods using the famous (Fisher's or Anderson's) iris data set. The data set gives the measurements in centimeters of the variables sepal length and width, and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are iris setosa, versicolor, and virginica.

```
# Scale Sepal.Length, Sepal.Width, Petal.Length and Petal.Width and drop Species
iris.scaled = as.data.frame(scale(iris[, -5]))
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
str(iris.scaled)
```

```
## 'data.frame':    150 obs. of  4 variables:
##  $ Sepal.Length: num  -0.898 -1.139 -1.381 -1.501 -1.018 ...
##  $ Sepal.Width : num  1.0156 -0.1315 0.3273 0.0979 1.245 ...
##  $ Petal.Length: num  -1.34 -1.34 -1.39 -1.28 -1.34 ...
##  $ Petal.Width : num  -1.31 -1.31 -1.31 -1.31 -1.31 ...
```

- **k-means clustering via `kmeans()`**

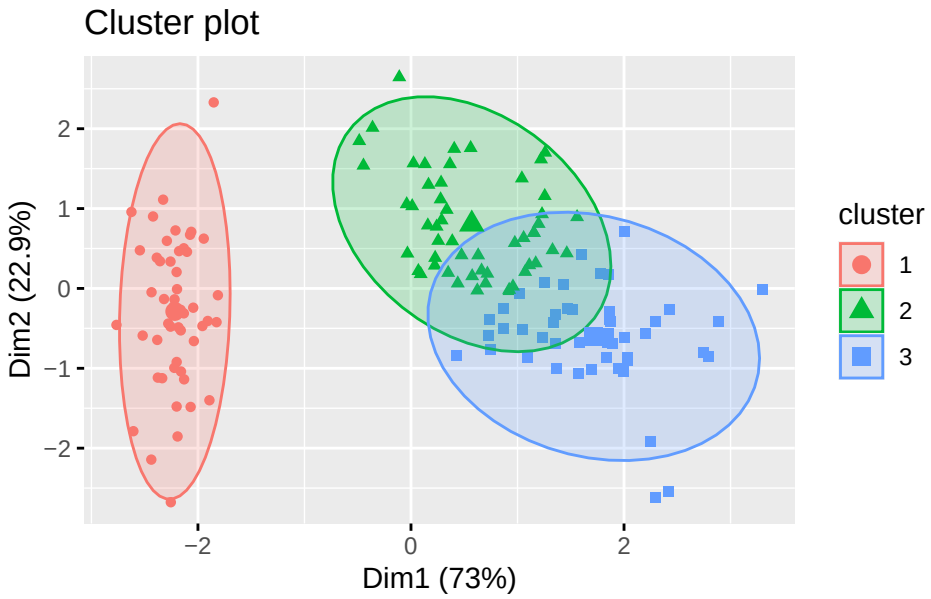
```
set.seed(123)
iris.km = kmeans(iris.scaled, centers=3, nstart=25)
# k-means group number of each observation
iris.km$cluster
```

```
##   [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 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 2 2 3 2 2 2 2 2 2 2 3 2 2 2 2 3 2 2 2
##  [75] 2 3 3 3 2 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 3 3 3 3 3 3 2 2 3 3 3 2 3 3 3 2 3 3 3 2 3
## [149] 3 2
```

Let's visualize k-means clusters by `fviz_cluster()` in package `factoextra`.

```
# install.packages("factoextra")
library(factoextra)

fviz_cluster(iris.km, data = iris.scaled, geom = "point",
             stand = FALSE, frame.type = "norm")
```



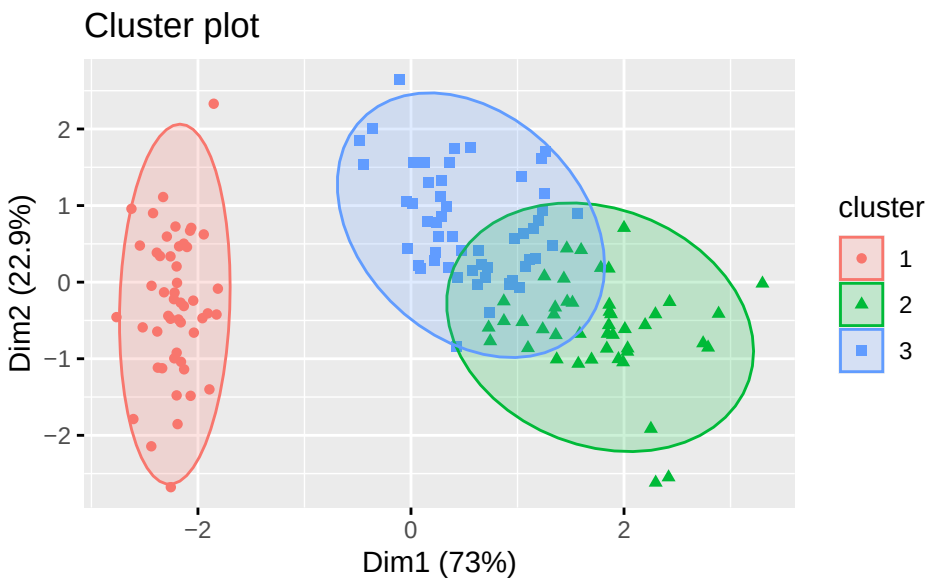
- **k-medioids clustering with `pam()`**

```
# install.pcakages("cluster")
library(cluster)
set.seed(321)
iris.pam = pam(iris.scaled, k=3, keep.diss=F)
iris.pam$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 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3
##  [75] 3 2 2 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 2 2 2 2 3 2 2 2 2
## [112] 2 2 3 2 2 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 2 3 3 2 2 2 3 2 2 2 3 2 2 2 3 2
## [149] 2 3
```

Visualize k-medioids clusters by `fviz_cluster()` too.

```
fviz_cluster(iris.pam, stand = FALSE, geom = "point",
             frame.type = "norm")
```

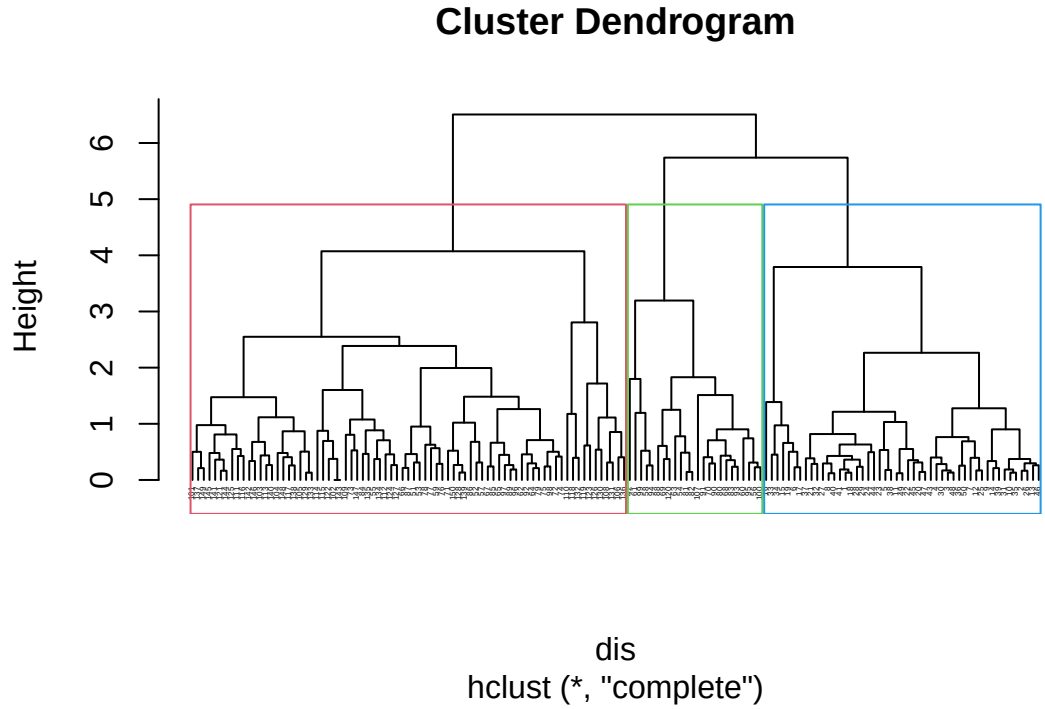


- **hierachical clusering with `hclust()`**

```
dis = dist(iris.scaled, method="euclidean")
set.seed(213)
iris.hc = hclust(dis, method="complete")
```

Plot dendrogram for hierachical clusering visualization.

```
plot(iris.hc, hang=-1, labels=, main='Cluster Dendrogram', cex=0.25)
# Add rectangle around 3 groups
rect.hclust(iris.hc, k=3, border = 2:4)
```



```r
# Cut the dendrogram in order to have 3 clusters
hc.cut = cutree(iris.hc, k=3)
hc.cut
```

```
##   [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 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 2 1 1 1 1 1 1 1 1 3 3 3 2 3 2 3 2 3 2 2 3 2 3 3 3 3 2 2 2 3 3 3 3
##  [75] 3 3 3 3 3 2 2 2 2 3 3 3 3 2 3 2 2 3 2 2 2 3 3 3 2 2 3 3 3 3 3 3 2 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

One thing we can observe from the above dendrogram is that it's hard to read the labels at the bottom because they would be either too small/too big to see. This necessitates us using a more powerful package for producing dendrograms.

2. Dendrogram by `dendextend`

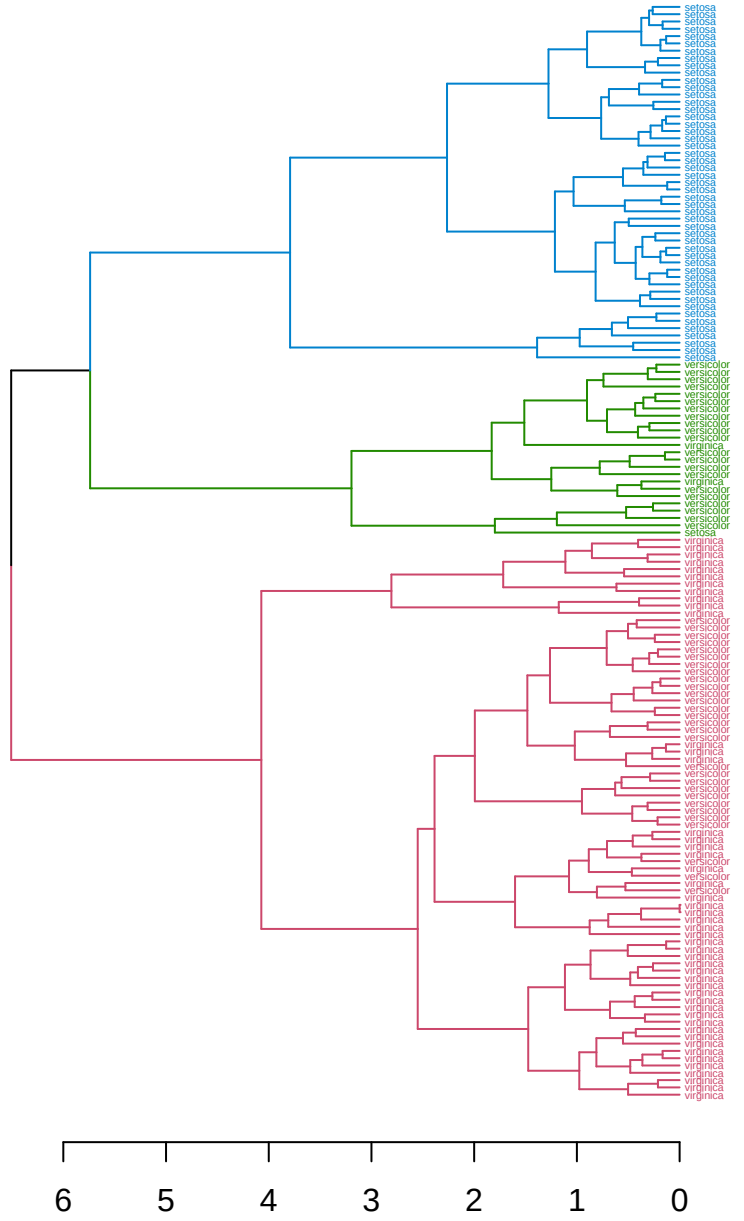
As we can see from the above results of cluster dendrogram, when the number of observations is large, the dendrogram looks messy, making it less interpretable. Here we introduce another R package called `dendextend`. First of all, the function `as.dendrogram()` is used to produce tree-like structures. Next, functions `color_branches()` and `color_labels()` can color all branches and labels by clusters. For example,

```r
# install.packages("dendextend")
library(dendextend)

## dendrogram: branches colored by 3 groups
dend1 = as.dendrogram(iris.hc)
# color branches and labels by 3 clusters
dend1 = color_branches(dend1, k=3)
dend1 = color_labels(dend1, k=3)
# change label size
dend1 = set(dend1, "labels_cex", 0.3)
```

Then, instead of having row indices as the branch labels (as seen in the previous plot), we use the true **species** of each observation to label the branches. In order to make the dendrogram more readable, we would like to rotate it counter-clockwise, making the tips shown on the right. Notice that here one should order the **species** by its clustering results.

```r
# add true labels to observations
dend1 = set_labels(dend1, labels=iris$Species[order.dendrogram(dend1)])
# plot the dendrogram
plot(dend1, horiz=T, main = "Dendrogram colored by three clusters")
```



Question

- How do we determine the optimal number of clusters?
- Can we perform the clustering analysis by other functions?
- How do we assess the performance of a clustering model?

Before we answer these questions, let's learn a new function/package `NbClust()` (in package `Nbclust`), which will help us simultaneously fit clustering model, computes multiple indices and determine the number of clusters in a single function call.

3. NbClust – A package providing 30 indices for determining the best number of clusters

Most of the clustering algorithms depend on some assumptions in order to define the subgroups present in a data set. As a consequence, the resulting clustering scheme requires some sort of evaluation as regards its validity. The evaluation procedure has to tackle difficult problems such as the quality of clusters, the degree with which a clustering scheme fits a specific data set and the optimal number of clusters in a partitioning.

The package `Nbclust`, published by Charrad et al., 2014[1], provides 30 indices for determining the relevant number of clusters, and proposes to users the best clustering scheme from the different results obtained by varying all combinations of number of clusters, distance measures, and clustering methods. In the current version of the `NbClust` package, only k-means and the agglomerative approach of hierarchical clustering are available.

An important advantage of `NbClust` is that the user can simultaneously computes multiple indices and determine the number of clusters in a single function call.

```
# install.packages("NbClust", repos = "http://cran.us.r-project.org")
library(NbClust)
```

The simplified format of the function **`NbClust`** is:

- **NbClust**(data = NULL, diss = NULL, distance = "euclidean", min.nc = 2, max.nc = 15, method = NULL, index = "all")
  - data: matrix
  - diss: dissimilarity matrix to be used. By default, diss=NULL, but if it is replaced by a dissimilarity matrix, distance should be "NULL"
  - distance: the distance measure to be used to compute the dissimilarity matrix. Possible values include "euclidean", "manhattan" or "NULL".
  - min.nc, max.nc: minimal and maximal number of clusters, respectively
  - method: The cluster analysis method to be used including "kmeans," "ward.D", "ward.D2", "single", "complete", "average" and more
  - index: the index to be calculated including "silhouette", "gap" and more.

(a) Determined the best number of clusters using only one index of interest

`NbClust()` makes our life a lot easier whenever we want to fit a kmeans/hierarchical clustering and determine the best number of clusters at the same time. For example, if we'd like to perform k-means and to use silhouette coefficient as the judging measure for the best number of clusters, we can do:

```
nb.km = NbClust(iris.scaled, distance = "euclidean",
                  min.nc = 2, max.nc = 10,
                  method = "kmeans", index ="silhouette")
nb.km
```

```
## $All.index
##      2      3      4      5      6      7      8      9     10
## 0.5818 0.4599 0.3869 0.3455 0.3266 0.3254 0.3227 0.3388 0.3377
##
## $Best.nc
## Number_clusters     Value_Index
```

[1]For more details, please read the paper *NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set* by Charrad et al. https://www.jstatsoft.org/article/view/v061i06/v61i06.pdf.

```
##          2.0000          0.5818
##
## $Best.partition
##   [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 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2
```

In the output, `$All.index` gives all silhouette coefficient values, `$Best.nc` displays the best number of clusters for the data set and its corresponding silhouette coefficient, and `$Best partition` shows the partitioned observations for `$Best.nc`.

Similarly, we can fit a hierarchical clustering with complete linkage by `NbClust()`. Here we use the gap index to determine the best number of clusters.

```
nb.hc = NbClust(iris.scaled, distance = "euclidean",
                 min.nc = 2, max.nc = 10,
                 method = "complete", index ="gap")
nb.hc
```

```
## $All.index
##       2       3       4       5       6       7       8       9      10
## -0.2899 -0.2303 -0.6915 -0.8606 -1.0506 -1.3223 -1.3303 -1.4759 -1.5551
##
## $All.CriticalValues
##       2       3       4       5       6       7       8       9      10
## -0.0539  0.4694  0.1787  0.2009  0.2848  0.0230  0.1631  0.0988  0.1708
##
## $Best.nc
## Number_clusters     Value_Index
##          3.0000         -0.2303
##
## $Best.partition
##   [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 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 2 1 1 1 1 1 1 1 1 3 3 3 2 3 2 3 2 3 2 2 3 2 3 3 3 3 2 2 2 3 3 3 3
##  [75] 3 3 3 3 3 2 2 2 2 3 3 3 3 2 3 2 2 3 2 2 2 3 3 3 2 2 3 3 3 3 3 3 2 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

This time, the best number of clusters is 3 and correspondingly the gap index is -0.2303.

(b) Determined the best number of clusters using multiple indices

To compute multiple indices simultaneously, the possible values for the argument index can be “alllong”" or “all”. The option “alllong” requires more time, as the run of some indices, such as Gamma, Tau, Gap and Gplus, is computationally very expensive. The user can avoid computing these four indices by setting the `index=all`. In this case, only 26 indices are calculated.

With the `index=alllong` option, the output of the `NbClust()` contains:

- all validation indices
- critical values for Duda, Gap, PseudoT2 and Beale indices
- the number of clusters corresponding to the optimal score for each index
- the best number of clusters proposed by NbClust according to the majority rule
- the best partition

Let's perform a hierarchical clustering with complete linkage function using `index=alllong` and `index=all` respectively.

```
# Select the best number of clusters by 30 indices
nb.hc.alllong = NbClust(iris.scaled, distance = "euclidean", min.nc = 2,
        max.nc = 10, method = "complete", index ="alllong")

# Select the best number of clusters by 27 indices
nb.hc.all = NbClust(iris.scaled, distance = "euclidean", min.nc = 2,
        max.nc = 10, method = "complete", index ="all")
```

Due to the length of this lab handout, we do not print out the outputted plots of `nb.hc.alllong` and `nb.hc.all`, but you have to look at them by yourself to understand the essence of selecting the best number of clusters by `Nbclust()`.

Certainly, the results presented in the example above seem to indicate that there is no unanimous choice regarding the optimal number of clusters. For instance, in the text output of `nb.hc.all`, Some indices propose 2 as the best number of clusters, some indices propose 3 as the optimal number of clusters, the rest indices select 10 as the relevant number of clusters in this data set. The package chooses the best cluster based on the majority vote. The optimal number of clusters would be 3, as it is selected by 18 indices, which coincides with 3 types of flowers.

(c) Some indices implemented in the NbClust package

Each index in `NbClust` has its own definition and way to select the optimal number of clusters. In the help file of `NbClust`, there is a full list of each index and its corresponding criterion. We list 3 of them for illustration. For more indices, please refer to the paper of Charrad et al.

Index in `NbClust`	Optimal number of clusters is chosen by
silhouette	Maximum value of the index
gap	Smallest number of clusters such that criticalValue ≥ 0
gamma	Maximum value of the index

4. Clustering validation

In the previous section, we determined the optimal number of clusters for iris dataset using different indices. This is a crucial aspect of cluster validation. In the cases of supervised classification, we have a variety of measures to evaluate how good our model is, such as MSE, accuracy, precision and recall. However, for cluster analysis, the analogous question is how to evaluate the "goodness" of the resulting clusters?

There are different aspects cluster validation:

- Determining the 'correct' number of clusters (as shown in section 2)
- Evaluating how well the results of a cluster analysis fit the data without reference to external information, i.e., use only the data
- Comparing the results of two different sets of cluster analyses to determine which is better
- Determining the clustering tendency of a set of data, i.e., distinguishing whether non-random structure actually exists in the data
- Comparing the results of a cluster analysis to externally known results, e.g., to externally given class labels.

(a) Numerical measures

Numerical measures can be applied to judge various aspects of cluster validity, simple examples are correlation, SSE, Jaccard index, Rand index or a simple table (we have seen a table used for comparing different clustering methods in Lab06) to compare different clustering solutions. The calculation of correlation and SSE should be easy for you. Now we show the way to compute Jaccard and Rand index: we use `comPart()` function in package `flexclust`. Both indices are a measure of the similarity between two clustering methods.

```
# install.packages("flexclust")
library(flexclust)
```

For example, use Jaccard and Rand index to compare PAM and k-means clustering results. Recall that the object of PAM was saved as `iris.pam` and that of k-means as `iris.km`:

```
# jaccard index
comPart(iris.pam$clustering, iris.km$cluster, type="J")
```

```
##         J
## 0.9011329
```

```
# rand index
comPart(iris.pam$clustering, iris.km$cluster, type="RI")
```

```
##        RI
## 0.9656376
```

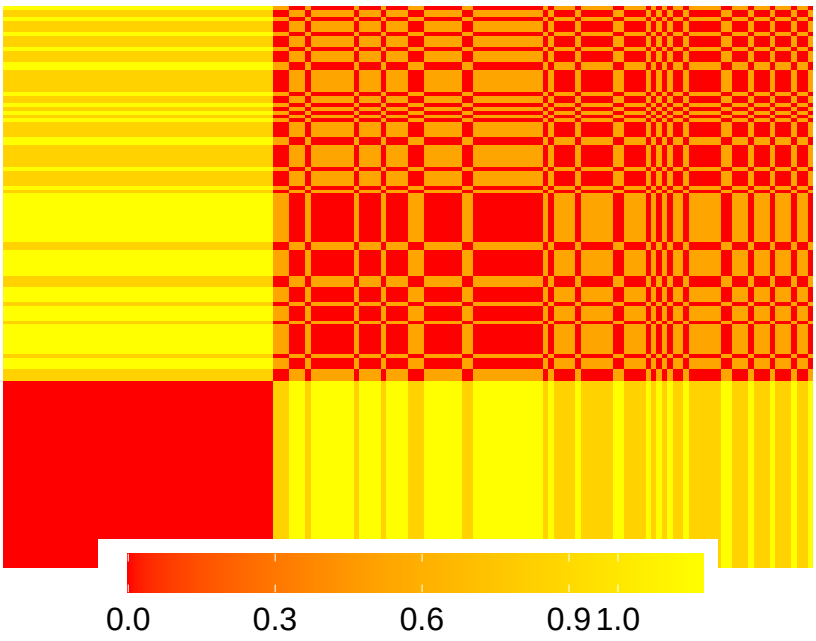
Similarly, you can compare `iris.pam` vs. `iris.hc`, and `iris.km` vs. `iris.hc`.

(b) Graphical measure for kmeans and hierarchical clustering: distance matrix visualization

```
library(superheat)

# recall that dis is the euclidean distance matrix obtained by
# dis = dist(iris.scaled, method="euclidean")

superheat(as.matrix(dis)[iris.km$cluster, iris.km$cluster],
          heat.pal = c("red", "orange", "yellow"))
```



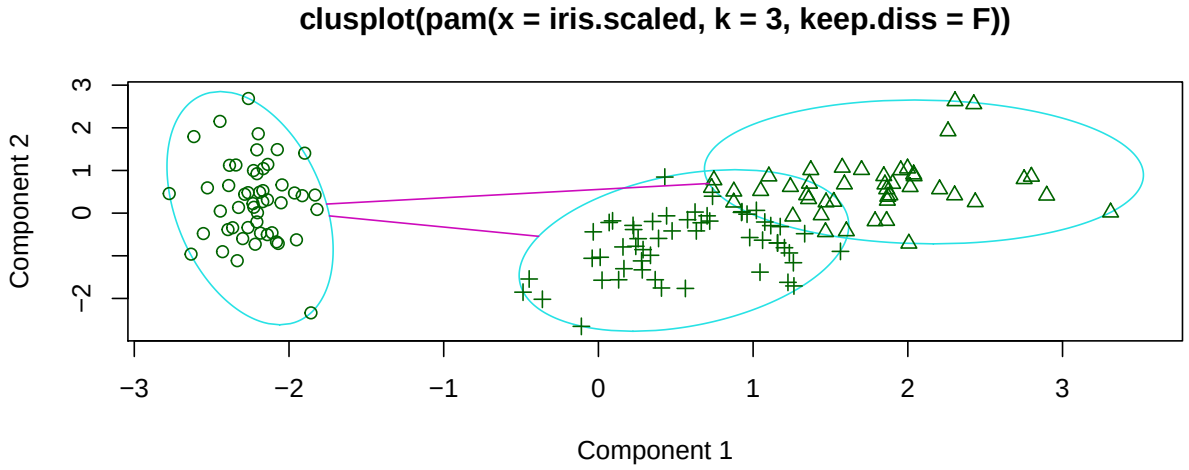
(c) Graphical measure for pam: Silhouette coefficient and Silhouette plot

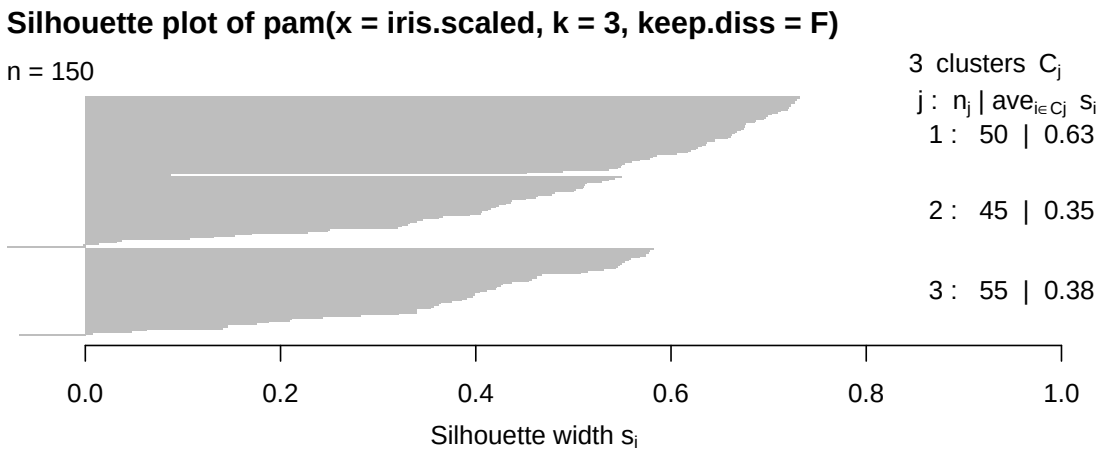
The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to 1. Values near 1 mean that the observation is well placed in its cluster; values near 0 mean that it's likely that an observation might really belong in some other cluster. Within each cluster, the value for this measure is displayed from smallest to largest. The *maximum value* of the Silhouette coefficient is used to determine the optimal number of clusters in the data. This is exactly the same rule in 2(c).

Let's compute silhouette information for pam clustering object `pam` using `silhouette` function.

```
# Silhouette coefficient
sil.pam = silhouette(iris.pam$cluster, dis)

# Silhouette plot
plot(iris.pam)
```





If the silhouette plot shows values close to 1 for each observation, the fit was good; if there are many observations closer to 0, it's an indication that the fit was not good. The silhouette plot is very useful in locating groups in a cluster analysis that may not be doing a good job; in turn this information can be used to help select the proper number of clusters.

In our plot, it indicates that there is a good structure to the clusters, with most observations seeming to belong to the cluster that they're in.

Note: The silhouette plot does not show correctly in R Studio if you have too many objects (bars are missing). It will work when you open a new plotting device with `windows()`, `x11()` or `quartz()`.

5. Your turn

Problem 1:

Use dendextend package to create a colored dendrogram by partitioning all observations into 5 clusters.

Problem 2:

Use NbClust to perform a hierachical clustering (single linkage, Manhattan distance) and choose the best

value of clusters using all indices

Credit: Adopted from https://www.jstatsoft.org/article/view/v061i06/v61i06.pdf, http://www.cs.kent.edu/~jin/DM08/ClusterValidation.pdf and http://www.sthda.com/english/wiki/print.php?id=239