Hierarchical Clustering

CITS4009 Computational Data Analysis

Unit Coordinator: Dr Du Huynh

Department of Computer Science and Software Engineering
The University of Western Australia

Semester 2, 2022

Clustering

Grouping countries based on protein consumption

A small dataset from 1973 on protein consumption

- 9 food groups, 25 countries in Europe
  https://github.com/WinVector/zmPDSwR/tree/master/Protein/

```
path <- '../../data_v2/Protein/'
df <- read.table(paste0(path,'protein.txt'), sep='\t', header=TRUE)
str(df)
## 'data.frame':    25 obs. of  10 variables:
##  $ Country  : Factor w/ 25 levels "Albania","Austria",..: 1 2 3 4 5 6 7 8 9 10 .
##  $ RedMeat  : num  10.1 8.9 13.5 7.8 9.7 10.6 8.4 9.5 18 10.2 ...
##  $ WhiteMeat: num  1.4 14 9.3 6 11.4 10.8 11.6 4.9 9.9 3 ...
##  $ Eggs     : num  0.5 4.3 4.1 1.6 2.8 3.7 3.7 2.7 3.3 2.8 ...
##  $ Milk     : num  8.9 19.9 17.5 8.3 12.5 25 11.1 33.7 19.5 17.6 ...
##  $ Fish     : num  0.2 2.1 4.5 1.2 2 9.9 5.4 5.8 5.7 5.9 ...
##  $ Cereals  : num  42.3 28 26.6 56.7 34.3 21.9 24.6 26.3 28.1 41.7 ...
##  $ Starch   : num  0.6 3.6 5.7 1.1 5 4.8 6.5 5.1 4.8 2.2 ...
##  $ Nuts     : num  5.5 1.3 2.1 3.7 1.1 0.7 0.8 1 2.4 7.8 ...
##  $ Fr.Veg   : num  1.7 4.3 4 4.2 4 2.4 3.6 1.4 6.5 6.5 ...
```

Units and Scaling

- *Units* (or more precisely, *disparity in units*) affect the clustering result, e.g.,
  - *height in feet* and *weight in pounds* will give different distances – and possibly different clusters – than if *height in meters* and *weight in kilograms* are used.
- Ideally, a unit of change in each coordinate should represent the same degree of difference
  - Animal-based food sources in general have more grams of protein per serving than plant-based food sources;
  - A change in consumption of 5 grams is a bigger difference in terms of vegetable consumption than it is in terms of red meat consumption.

Make clustering more coordinate-free

A general practice is to transform all the columns to have a mean value of 0 and a standard deviation of 1.

平均值为0，标准差为1

The `scale()` function in R

- returns a matrix as output, and
- annotates its output with two attributes.

```
vars.to.use <- colnames(df)[-1]  # all column names except for the 1st column
scaled_df <- scale(df[,vars.to.use]) # this is a 25-by-9 matrix where each
                                     # column has 0 mean and unit standard deviation
```

Make clustering more coordinate-free (cont.)

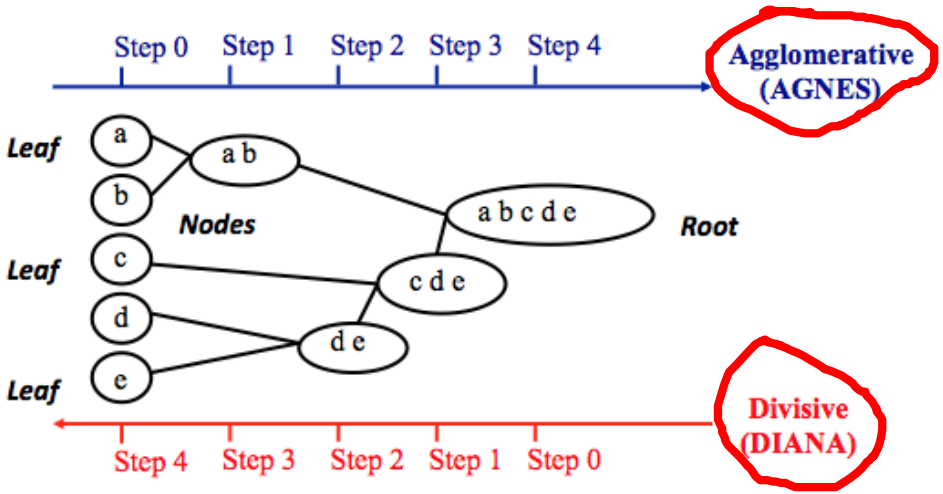
```
# The scaled:center attribute (a vector of 9 elements) contains the mean values
# of all the columns.
attr(scaled_df, "scaled:center")
##    RedMeat WhiteMeat       Eggs       Milk       Fish    Cereals     Starch       Nuts
##      9.828      7.896      2.936     17.112      4.284     32.248      4.276      3.072
##     Fr.Veg
##      4.136

# The scaled:scale attribute (a vector of 9 elements) contains the variances
# of all the columns.
attr(scaled_df, "scaled:scale")
##    RedMeat WhiteMeat       Eggs       Milk       Fish    Cereals     Starch       Nuts
##  3.347078   3.694081   1.117617   7.105416   3.402533 10.974786   1.634085   1.985682
##     Fr.Veg
##   1.803903
```

```
# The following also work:
attributes(scaled_df)$`scaled:center`
attributes(scaled_df)$`scaled:scale`
```

Hierarchical Clustering

Intuition Behind Hierarchical Clustering

We can follow either the **agglomerative** (bottom-up) or **divisive** (top-down) way of clustering.

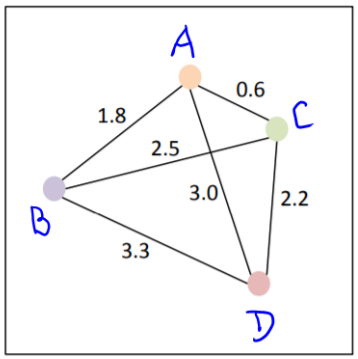


- **Agglomerative clustering** will start with n clusters, where n is the number of observations, and then group the most similar ones together.
- **Divisive clustering** will go the other way around – assuming all your n data points are one big cluster, and then dividing most dissimilar ones into separate groups.

Hierarchical Clustering Illustration

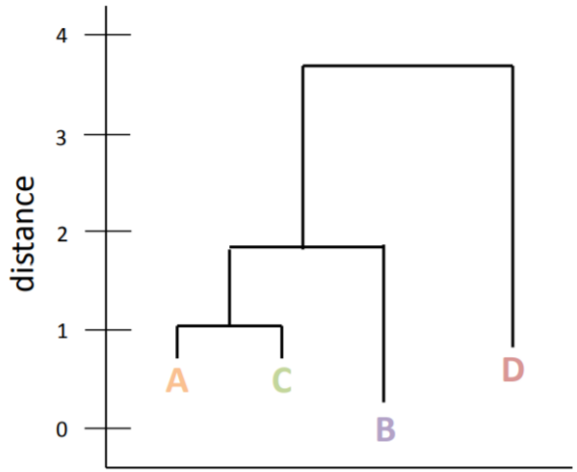
	A	B	C	D	...
A	0	1.8	0.6	3.0	
B	1.8	0	2.5	3.3	
C	0.6	2.5	0	2.2	
D	3.0	3.3	2.2	0	
...					

4*3/2=6



If distances are not equal between points we can draw a “hanging tree” to illustrate distances

The result of a cluster analysis is a tree or dendrogram



Methods for grouping

从距离最短的一对点开始

- **Nearest Neighbour Method** – creates groups by starting with points that are close together (having smallest distances) and build branches
  - In effect we keep asking the data matrix "Which point is my nearest neighbour?" to add branches

从离中心最近的点开始

- **Centroid Method** – creates a group based on points having the smallest distance to the group centroid rather than group member
  - First creates a group based on small distance then uses the centroid of that group to find which additional points belong in the same group
- **Ward's Method** – creates groups such that variance is minimised within clusters
  - Looks for spherical clusters

创建组，使集群内的差异最小化

Hierarchical Clustering using `hclust()`

The `hclust()` function:

- Input: a distance matrix (an object of class `dist`), which records the distances between all pairs of points in the data (using any one of the distance metrics).
- Output: a *dendrogram* - a tree that represents the nested clusters.
- A variety of clustering methods can be used to produce a tree that records the nested cluster structure.

树状图

Compute the pairwise distances using `dist()`

`dist()` will calculate distance functions using

- the (squared) Euclidean distance (`method="euclidean"`),
- the Manhattan distance (`method="manhattan"`), and
- something like the Hamming distance, when categorical variables are expanded to indicators (`method="binary"`).

If you want to use another distance metric, you'll have to compute the appropriate distance matrix and convert it to a dist object using the `as.dist()`.

The output of dist() function

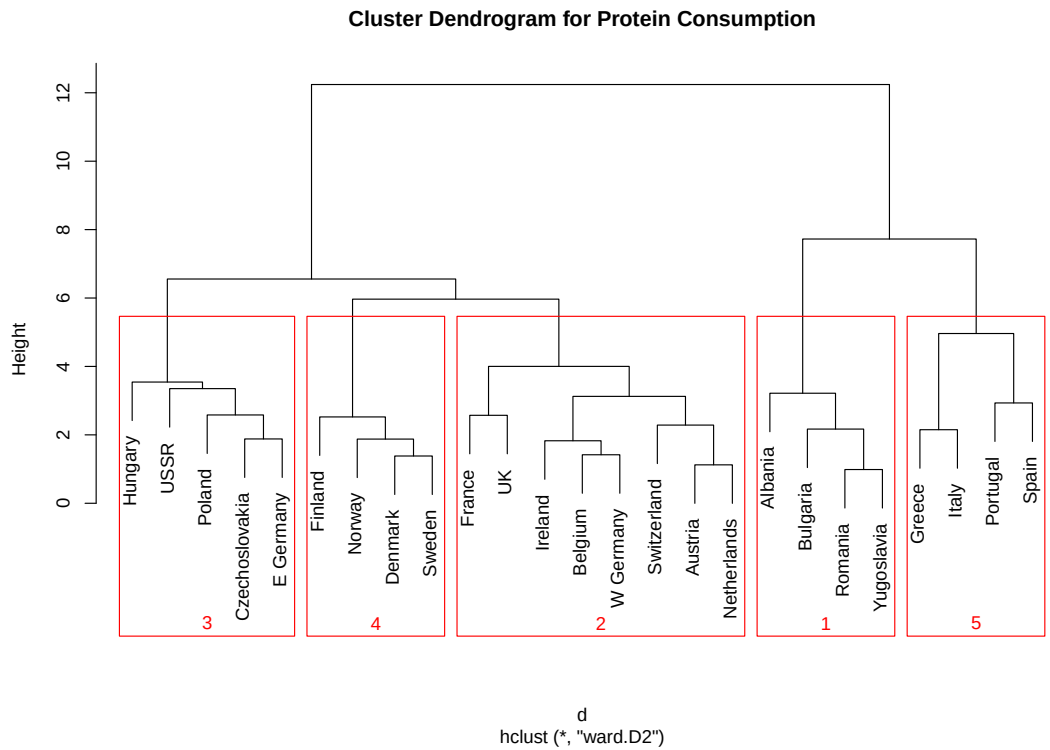
```
# dist() returns an object of class 'dist', which includes the pairwise
# Euclidean distances of the 25 observations. In this case, we have
# 25-choose-2 pairs (i.e., 25*24/2 = 300). So 300 Euclidean distance values
# are stored in variable `d` for further operation.
(d <- dist(scaled_df, method="euclidean"))
```

```
##           1        2        3        4        5        6        7
## 2  6.1360510
## 3  5.9487614 2.4513223
## 4  2.7645367 4.9002772 5.2370880
## 5  5.1411480 2.1308906 2.2202549 3.9503
## 6  6.6341625 3.0192376 2.5391149 6.0365
## 7  6.3922502 2.5819811 2.1140954 5.4091332 1.8802660 2.7645740
## 8  5.8794649 4.0716662 3.5042937 5.8176296 3.9839792 2.6334871 4.0723681
## 9  6.3070106 3.5889373 2.1944157 5.5606801 3.3690903 3.6628201 3.7933343
## 10 4.2559301 5.1639348 4.6951531 3.7626971 4.8701393 5.5968794 5.6196014
## 11 4.6734470 3.2857392 3.9946781 3.3453809 2.7513287 5.0390826 3.6762821
## 12 6.7571210 2.7408193 1.6567570 6.2125897 3.1576644 2.8295563 3.0335760
## 13 4.0255665 3.7175506 3.7186963 2.8654638 3.3461638 4.7794027 4.3204290
## 14 6.0080723 1.1233122 2.2489670 5.1685911 2.1938164 2.5365988 2.5318280
## 15 5.4652307 3.8755033 2.9606943 5.2768296 3.5270340 1.9936668 3.2723624
## 16 5.8828058 2.7959971 2.9359042 4.4345191 2.1044427 3.8446943 2.7089165
## 17 6.6120099 6.5291699 5.6512736 6.0046278 5.5189793 5.8699772 5.2526342
## 18 2.6895968 4.6504990 4.7603534 1.8894354 3.5622341 5.5339127 4.7841740
```

The output from the function is a lower triangular matrix (as the matrix is symmetric so only half of the matrix is needed.)

Clustering using hclust()

```
pfit <- hclust(d, method="ward.D2")  # perform hierarchical clustering
# To examine `pfit`, type: summary(pfit)   and   pfit$height
plot(pfit, labels=df$Country, main="Cluster Dendrogram for Protein Consumption")
rect.hclust(pfit, k=5)   # k=5 means we want rectangles to be put around 5 clusters
xx <- c(3, 7.5, 13.5, 19.5, 23.5); yy <- -3.5; clusterID <- c(3,4,2,1,5)
text(xx, yy, clusterID, col="red")
```



Extract members of each cluster using `cutree()`

`cutree()` takes in a clustering model (e.g., created by `hclust()`) and returns a vector (or matrix) of cluster group assignment(s) for each row. We can specify the desired number(s) of groups (using the keyword argument `k`) or the desired height value(s) (using the keyword argument `h`) where the tree should be cut.

k：所需的组数
h：所需的树的高度

```
groups <- cutree(pfit, k=5)
groups
##  [1] 1 2 2 1 3 4 3 4 2 5 3 2 5 2 4 3 5 1 5 4 2 2 3 2 1

print_clusters <- function(df, groups, cols_to_print) {
  Ngroups <- max(groups)
  for (i in 1:Ngroups) {
    print(paste("cluster", i))
    print(df[groups == i, cols_to_print])
  }
}
```

Extract members of each cluster using cutree() (cont.)

```
cols_to_print <- c("Country","RedMeat","Fish","Fr.Veg")
print_clusters(df, groups, cols_to_print)
## [1] "cluster 1"
##       Country RedMeat Fish Fr.Veg
## 1     Albania    10.1  0.2    1.7
## 4    Bulgaria     7.8  1.2    4.2
## 18    Romania     6.2  1.0    2.8
## 25 Yugoslavia     4.4  0.6    3.2
## [1] "cluster 2"
##        Country RedMeat Fish Fr.Veg
## 2      Austria     8.9  2.1    4.3
## 3      Belgium    13.5  4.5    4.0
## 9       France    18.0  5.7    6.5
## 12     Ireland    13.9  2.2    2.9
## 14 Netherlands     9.5  2.5    3.7
## 21 Switzerland    13.1  2.3    4.9
## 22          UK    17.4  4.3    3.3
## 24   W Germany    11.4  3.4    3.8
## [1] "cluster 3"
##           Country RedMeat Fish Fr.Veg
## 5  Czechoslovakia     9.7  2.0    4.0
```

Grouping interpretation

We observe that the countries in each cluster tend to be in the same geographical region.

It makes sense that countries in the same region would have similar dietary habits. You can also see that

- Cluster 2 contains countries with higher-than-average red meat consumption.
- Cluster 4 contains countries with higher-than-average fish consumption but low produce consumption.
- Cluster 5 contains countries with high fish and produce consumption.

我们观察到，每个集群中的国家往往位于同一地理区域：集群2包含红肉消费量高于平均水平的国家。类群4包含鱼类消费量高于平均水平但农产品消费量较低的国家。类组5包含鱼类和农产品消费量高的国家

Visualising Clusters - Data preparation

```
princ <- prcomp(scaled_df) # Calculate the principal components of scaled_df
nComp <- 2         # focus on the first two principal components
# project scaled_df onto the first 2 principal components to form a new
# 2-column data frame.
project2D <- as.data.frame(predict(princ, newdata=scaled_df)[,1:nComp])
# combine with `groups` and df$Country to form a 4-column data frame

hclust.project2D <- cbind(project2D, cluster=as.factor(groups), country=df$Country)
head(hclust.project2D)
##           PC1         PC2 cluster        country
## 1  1.7038269 -0.83620210       1        Albania
## 2 -0.6985798 -0.39055549       2        Austria
## 3 -0.8214716  0.07628159       2        Belgium
## 4  1.1679022 -0.49288194       1       Bulgaria
## 5 -0.2565371 -0.16947019       3 Czechoslovakia
## 6 -0.8804527 -0.07393348       4        Denmark
```

PCA:principal component analysis

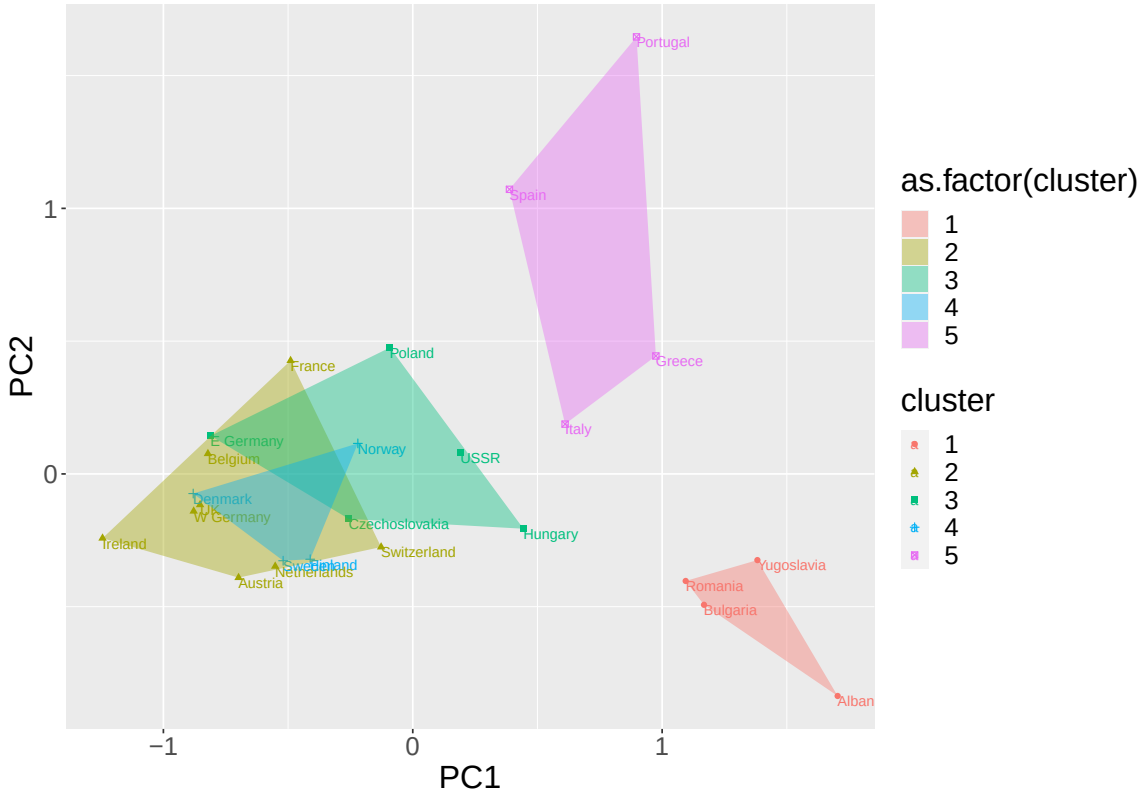
Visualising Clusters - Finding the convex hull

```
# finding convex hull
library('grDevices')
find_convex_hull <- function(proj2Ddf, groups) {
  do.call(rbind,
          lapply(unique(groups),
                 FUN = function(c) {
                   f <- subset(proj2Ddf, cluster==c);
                   f[chull(f),]
                 }
                )
          )
}
hclust.hull <- find_convex_hull(hclust.project2D, groups)
```

`chull(x, y = NULL)` computes the convex hull of a given set of points. It returns a vector of indices of the unique points lying on the convex hull.

Visualising Clusters using ggplot

```
library(ggplot2)
ggplot(hclust.project2D, aes(x=PC1, y=PC2)) +
    geom_point(aes(shape=cluster, color=cluster)) +
    geom_text(aes(label=country, color=cluster), hjust=0, vjust=1, size=3) +
    geom_polygon(data=hclust.hull, aes(group=cluster, fill=as.factor(cluster)),
                 alpha=0.4, linetype=0) + theme(text=element_text(size=20))
```



Assess whether a cluster represents true structure

To see if the cluster holds up under plausible variations in the dataset, `clusterboot()` from the `fpc` package uses bootstrap resampling to evaluate how stable a given cluster is.

- `clusterboot()` is an integrated function that both performs the clustering and evaluates the final produced clusters.
- It has interfaces to a number of R clustering algorithms, including both `hclust` and `kmeans`.
- `clusterboot()` uses the **Jaccard coefficient**, a similarity measure between sets A and B defined to be *the ratio of the number of elements in the intersection of A and B over the number of elements in the union of A and B*:

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

kmeans：（1）随机选取k个质心（k值取决于你想聚成几类）（2）计算样本到质心的距离，距离质心距离近的归为一类，分为k类（3）求出分类后的每类的新质心（4）再次计算计算样本到新质心的距离，距离质心距离近的归为一类（5）判断新旧聚类是否相同，如果相同就代表已经聚类成功，如果没有就循环2-4步骤直到相同

The basic general strategy

1. Cluster the data as usual.
2. Draw a new dataset (of the same size as the original) by *resampling* the original dataset *with replacement* (meaning that some of the data points may show up more than once, and others not at all). Cluster the new dataset.
3. For every cluster in the original clustering, find the most similar cluster in the new clustering (the one that gives the maximum Jaccard coefficient) and record that value.
   - If this maximum Jaccard coefficient is less than 0.5, then the original cluster is considered to be dissolved – it didn't show up in the new clustering.
   - A cluster that's dissolved too often is probably not a *real* cluster.
4. Repeat steps 2-3 several times.

Judging the stability of a clustering algorithm

The *clustering* stability of each cluster in the original clustering is the *mean value of its Jaccard coefficient* over all the bootstrap iterations.

- *Unstable* if stability < 0.6
- *Patterns are discovered but with low certainty* if $0.6 \leq$ stability ≤ 0.75
- *Highly stable possible real clusters* if stability > 0.85

Running `clusterboot()`

As different clustering algorithms can give different stability values, even when the algorithms produce highly similar clusterings, `clusterboot()` can be used to find out how stable the clustering algorithm is.

```
library(fpc)
kbest.p <- 5
cboot.hclust <- clusterboot(scaled_df, clustermethod=hclustCBI,
                            method="ward.D2", k=kbest.p)
```

```
## boot 1
## boot 2
## boot 3
## boot 4
## boot 5
## boot 6
## boot 7
## boot 8
## boot 9
## boot 10
## boot 11
## boot 12
## boot 13
```

Running `clusterboot()` (cont.)

We can use `summary()` to inspect the result from `clusterboot()`:

```
summary(cboot.hclust$result)
##                Length Class  Mode
## result          7     hclust list
## noise           1     -none- logical
## nc              1     -none- numeric
## clusterlist     5     -none- list
## partition      25     -none- numeric
## clustermethod   1     -none- character
## nccl            1     -none- numeric
```

Running clusterboot() (cont.)

```
groups.cboot <- cboot.hclust$result$partition
print_clusters(df, groups.cboot, "Country")
```

```
## [1] "cluster 1"
## [1] Albania    Bulgaria   Romania    Yugoslavia
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
## [1] "cluster 2"
## [1] Austria     Belgium     France      Ireland     Netherlands Switzerland
## [7] UK          W Germany
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
## [1] "cluster 3"
## [1] Czechoslovakia E Germany      Hungary        Poland         USSR
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
## [1] "cluster 4"
## [1] Denmark Finland Norway  Sweden
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
## [1] "cluster 5"
## [1] Greece   Italy    Portugal Spain
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
```

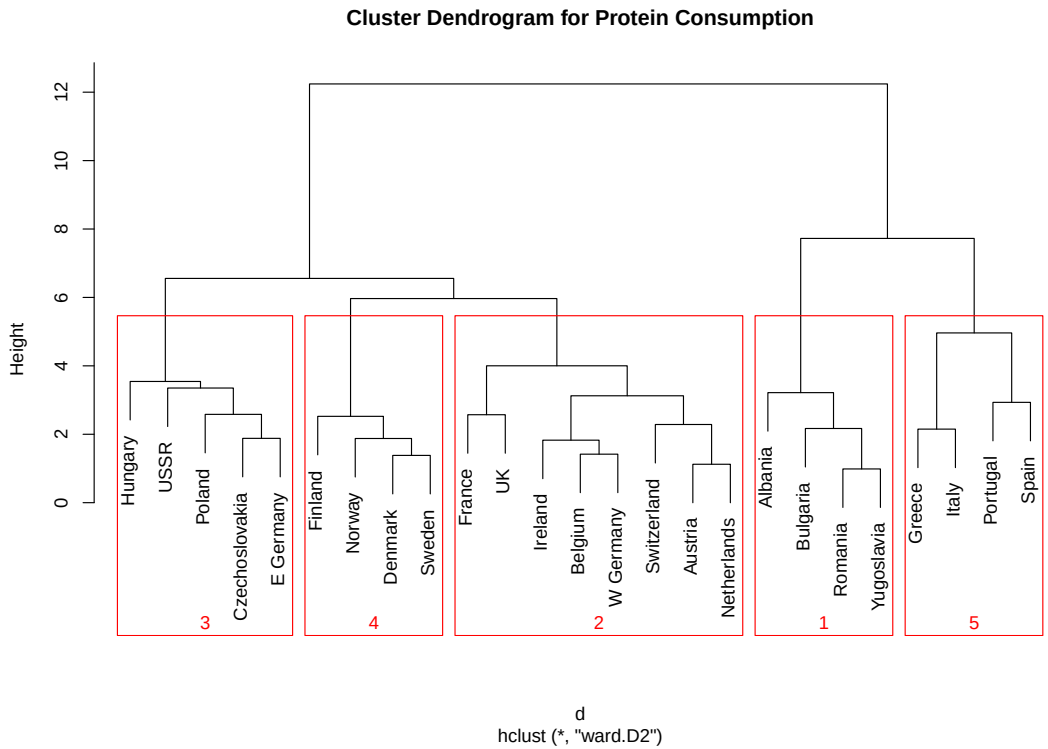
Count how many times each cluster was dissolved

By default, `clusterboot()` runs 100 bootstrap iterations.

```
1 - cboot.hclust$bootbrd/100
```

```
## [1] 0.84 0.86 0.55 0.80 0.66
```

So clusters 2 and 4 are highly stable.



Finding the number of clusters k

One simple heuristic for assessing the clustering results is to compute the total *within sum of squares* (WSS) for different values of k and look for an *elbow* in the curve.

The *centroid* of a cluster j with m elements is the mean value of all the points x_i in the cluster:

$$c_j = \frac{1}{m} \sum_{i=1}^{m} x_i$$

cj是第j个聚类中所有xi的平均值

所有xi与cj的距离平方的均值是wssj

1. Calculate the WSS:

The WSS value of a cluster is the *average squared distance of each point in the cluster from the cluster centroid*, e.g., the WSS of the j^{th} cluster is:

$$\text{WSS}_j = \frac{1}{m} \sum_{i=1}^{m} \| x_i - c_j \|^2$$

where $\| \cdot \|^2$ denotes the squared Euclidean distance.

Finding the number of clusters k (cont.)

2. Calculate the Total WSS:

If the total number of clusters is k then the total WSS is simply:

$$\text{total WSS} = \sum_{i=1}^{k} \text{WSS}_i$$

各个聚类的wss相加得到总的wss

The total WSS value will decrease as the number of clusters increases, because each cluster will be smaller and tighte

总WSS值将随着集群数量的增加而减少，因为每个集群将更小、更紧密

The hope is that the rate at which the WSS decreases will slow down for k beyond the optimal number of clusters.

我们希望WSS的下降速率在超过最佳簇数k时将会减慢

In other words, the graph of WSS versus k should flatten out beyond the optimal k, so the optimal k will be at the *elbow* of the graph.

最优k将在曲线肘部

Finding the number of clusters k (cont.)

3. Calculate the Calinski-Harabasz index:

The *Calinski-Harabasz index* (or CH index, for short) is another commonly used measure of cluster goodness.
To compute the CH index, we need to compute the following:

- The *total sum of squares* (TSS) – the sum of the squared distances of all the points from the centroid of the data. TSS is independent of the clustering. Given the centroid $\bar{x}$ and for n data points:

$$\text{TSS} = \sum_{i=1}^{n} (x_i - \bar{x})^2$$

.

- The *between sum of squares* (*BSS*), which measures how far apart the clusters are from each other, is defined as $\text{BSS} = \text{TSS} - \text{WSS}$. A good clustering has a small WSS (all the clusters are tight around their centres) and a large BSS.

Finding the number of clusters k (cont.)

3. Calculate the Calinski-Harabasz index (cont.):

The *within-cluster variance* W is given by:

$$\mathrm{W} = \mathrm{WSS}/(n-k)$$

where n is the total number of data points and k is the number of clusters.
The *between-cluster variance* B is given by:

$$\mathrm{B} = \mathrm{BSS}/(k-1)$$

Finding the number of clusters k (cont.)

一个好的聚类应该有一个大的B和一个小的W，所以我们应该尽量最大化CH指数

3. Calculate the Calinski-Harabasz index (cont.):

The Calinski-Harabasz (CH) index is defined as:

$$CH = B/W$$

A good clustering should have a large B and a small W so we should try to m

We can choose the CH index to find the k-value, which measures the tightness within a class by calculating the sum of the squares of the distances between the points in the class and the class centre, and the separation of the data set by calculating the sum of the squares of the distances between the centroids of each class and the centroids of the data set, with the CH index being obtained from the ratio of separation to tightness. Thereby, a larger CH represents a tighter class itself and a more dispersed class to class

Finding k – Implementation in R

```r
# Function to return the squared Euclidean distance of two given points x and y
sqr_euDist <- function(x, y) {
    sum((x - y)^2)
}

# Function to calculate WSS of a cluster, represented as a n-by-d matrix
# (where n and d are the numbers of rows and columns of the matrix)
# which contains only points of the cluster.
wss <- function(clustermat) {
    c0 <- colMeans(clustermat)
    sum(apply( clustermat, 1, FUN=function(row) {sqr_euDist(row, c0)} ))
}

# Function to calculate the total WSS. Argument `scaled_df`: data frame
# with normalised numerical columns. Argument `labels`: vector containing
# the cluster ID (starting at 1) for each row of the data frame.
wss_total <- function(scaled_df, labels) {
    wss.sum <- 0
    k <- length(unique(labels))
    for (i in 1:k)
        wss.sum <- wss.sum + wss(subset(scaled_df, labels == i))
    wss.sum
}
```

Finding k – Implementation in R (cont.)

```
# Function to calculate total sum of squared (TSS) distance of data
# points about the (global) mean. This is the same as WSS when the
# number of clusters (k) is 1.
tss <- function(scaled_df) {
   wss(scaled_df)
}
```

```
# Function to return the CH indices computed using hierarchical
# clustering (function `hclust`) or k-means clustering (`kmeans`)
# for a vector of k values ranging from 1 to kmax.
CH_index <- function(scaled_df, kmax, method="kmeans") {
    if (!(method %in% c("kmeans", "hclust")))
        stop("method must be one of c('kmeans', 'hclust')")

    npts <- nrow(scaled_df)
    wss.value <- numeric(kmax) # create a vector of numeric type
    # wss.value[1] stores the WSS value for k=1 (when all the
    # data points form 1 large cluster).
    wss.value[1] <- wss(scaled_df)
```

Finding k – Implementation in R (cont.)

Function CH_index continuing below...

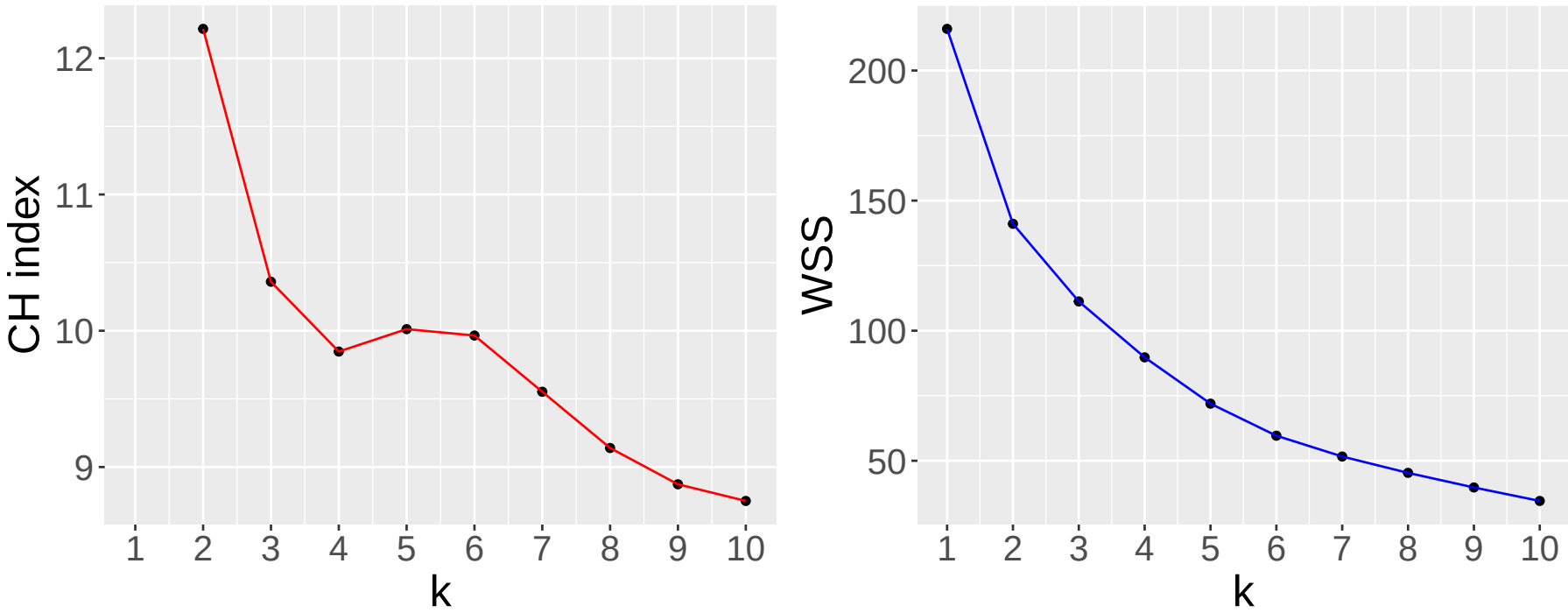
```
    if (method == "kmeans") {
        # kmeans
        for (k in 2:kmax) {
            clustering <- kmeans(scaled_df, k, nstart=10, iter.max=100)
            wss.value[k] <- clustering$tot.withinss
        }
    } else {
        # hclust
        d <- dist(scaled_df, method="euclidean")
        pfit <- hclust(d, method="ward.D2")
        for (k in 2:kmax) {
            labels <- cutree(pfit, k=k)
            wss.value[k] <- wss_total(scaled_df, labels)
        }
    }
    bss.value <- tss(scaled_df) - wss.value   # this is a vector
    B <- bss.value / (0:(kmax-1))             # also a vector
    W <- wss.value / (npts - 1:kmax)          # also a vector

    data.frame(k = 1:kmax, CH_index = B/W, WSS = wss.value)
}
```

Code to plot the CH index vs k

```
library(gridExtra)

# calculate the CH criterion
crit.df <- CH_index(scaled_df, 10, method="hclust")

fig1 <- ggplot(crit.df, aes(x=k, y=CH_index)) +
  geom_point() + geom_line(colour="red") +
  scale_x_continuous(breaks=1:10, labels=1:10) +
  labs(y="CH index") + theme(text=element_text(size=20))

fig2 <- ggplot(crit.df, aes(x=k, y=WSS), color="blue") +
  geom_point() + geom_line(colour="blue") +
  scale_x_continuous(breaks=1:10, labels=1:10) +
  theme(text=element_text(size=20))
```

Code to plot the CH index vs k (cont.)

```
grid.arrange(fig1, fig2, nrow=1)
```



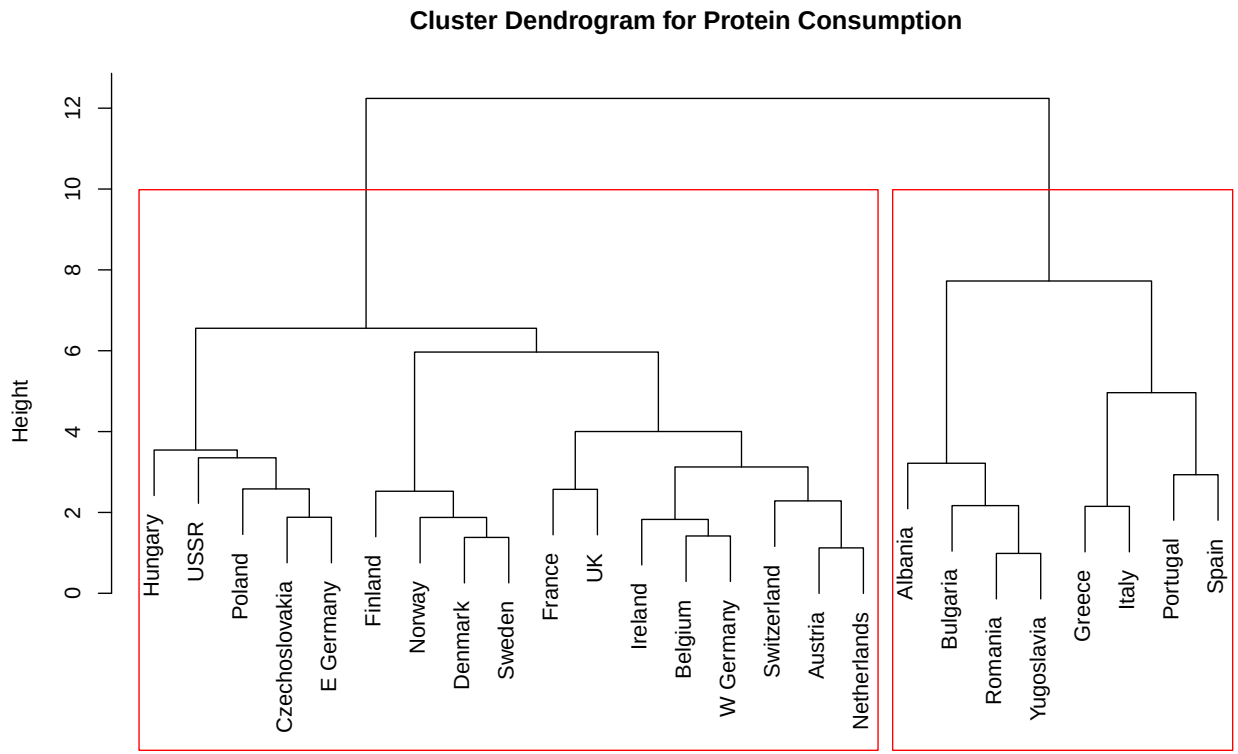
Looking at the figure, we see that the CH criterion is maximised at $k = 2$, with another local maximum at $k = 5$.

Output from `hclust()` with $k = 2$

Using $k = 2$, the output from `hclust()` is:

(Note that we don't need to recompute `pfit <- hclust(d, method="ward.D2")`)

```
plot(pfit, labels=df$Country, main="Cluster Dendrogram for Protein Consumption")
rect.hclust(pfit, k=2)
```



References

- Practical Data Science with R. By Nina Zumel and John Mount, Manning, 2014. (Chapter 8)
- Gower Dissimilarity: https://towardsdatascience.com/hierarchical-clustering-on-categorical-data-in-r-a27e578f2995
- Clustering Slides from Uni. of Alberta: https://sites.ualberta.ca/~lkgray/uploads/7/3/6/2/7362679/slides-clusteranalysis.pdf

kMeans Clustering

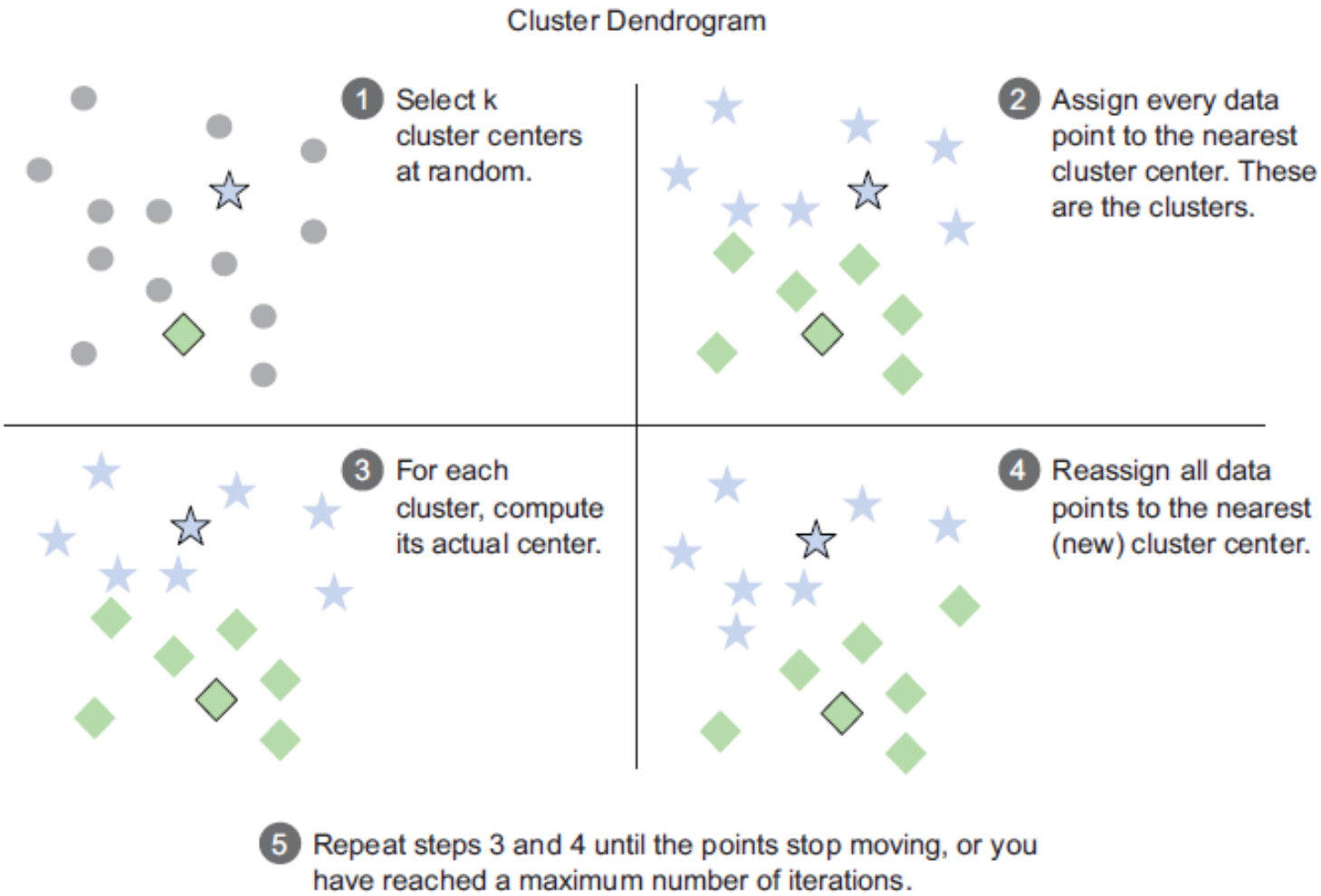
CITS4009 Computational Data Analysis

Unit Coordinator: Dr Du Huynh

Department of Computer Science and Software Engineering
The University of Western Australia

Semester 2, 2022

kMeans procedure



The `kmeans()` function

The function to run *k-means* in R is `kmeans()`. The output of the function includes :

- the cluster labels;
- the centres (centroids) of the clusters;
- the total sum of squares, total WSS, total BSS, and the WSS of each cluster.

```
kbest.p <- 5
# run kmeans with 5 clusters, 100 random starts, and 100
# maximum iterations per run.
kmClusters <- kmeans(scaled_df, kbest.p, nstart=100, iter.max=100)
```

Centres and sizes of the 5 Clusters

```
kmClusters$centers
##        RedMeat  WhiteMeat        Eggs       Milk       Fish    Cereals
## 1  0.006572897 -0.2290150  0.19147892  1.3458748  1.1582546 -0.8722721
## 2 -0.570049402  0.5803879 -0.08589708 -0.4604938 -0.4537795  0.3181839
## 3 -0.807569986 -0.8719354 -1.55330561 -1.0783324 -1.0386379  1.7200335
## 4 -0.508801956 -1.1088009 -0.41248496 -0.8320414  0.9819154  0.1300253
## 5  1.011180399  0.7421332  0.94084150  0.5700581 -0.2671539 -0.6877583
##       Starch       Nuts      Fr.Veg
## 1  0.1676780 -0.9553392 -1.11480485
## 2  0.7857609 -0.2679180  0.06873983
## 3 -1.4234267  0.9961313 -0.64360439
## 4 -0.1842010  1.3108846  1.62924487
## 5  0.2288743 -0.5083895  0.02161979
kmClusters$size
## [1] 4 5 4 4 8
cat("Total of cluster sizes =", sum(kmClusters$size))
## Total of cluster sizes = 25
cat("Total number of observations =", nrow(df))
## Total number of observations = 25
```

Extracted groups

```
groups <- kmClusters$cluster
print_clusters(df, groups, "Country")
## [1] "cluster 1"
## [1] Denmark Finland Norway  Sweden
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
## [1] "cluster 2"
## [1] Czechoslovakia E Germany      Hungary        Poland          USSR
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
## [1] "cluster 3"
## [1] Albania    Bulgaria   Romania    Yugoslavia
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
## [1] "cluster 4"
## [1] Greece   Italy    Portugal Spain
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
## [1] "cluster 5"
## [1] Austria     Belgium     France      Ireland     Netherlands Switzerland
## [7] UK          W Germany
## 25 Levels: Albania Austria Belgium Bulgaria Czechoslovakia ... Yugoslavia
```

`kmeansruns()` for picking k

We can use the `CH_index()` function to find the optimal k value that maximises the CH index. Alternatively, we can use `kmeansruns()` (from the `fpc` package) which calls `kmeans()` over a range of k and estimates the best k.

`kmeansruns()` has two criteria:

- The Calinski-Harabasz Index ("ch"), and
- The average silhouette width ("asw")

It's a good idea to plot the criterion values over the entire range of k, since you may see evidence for a k that the algorithm didn't automatically pick.

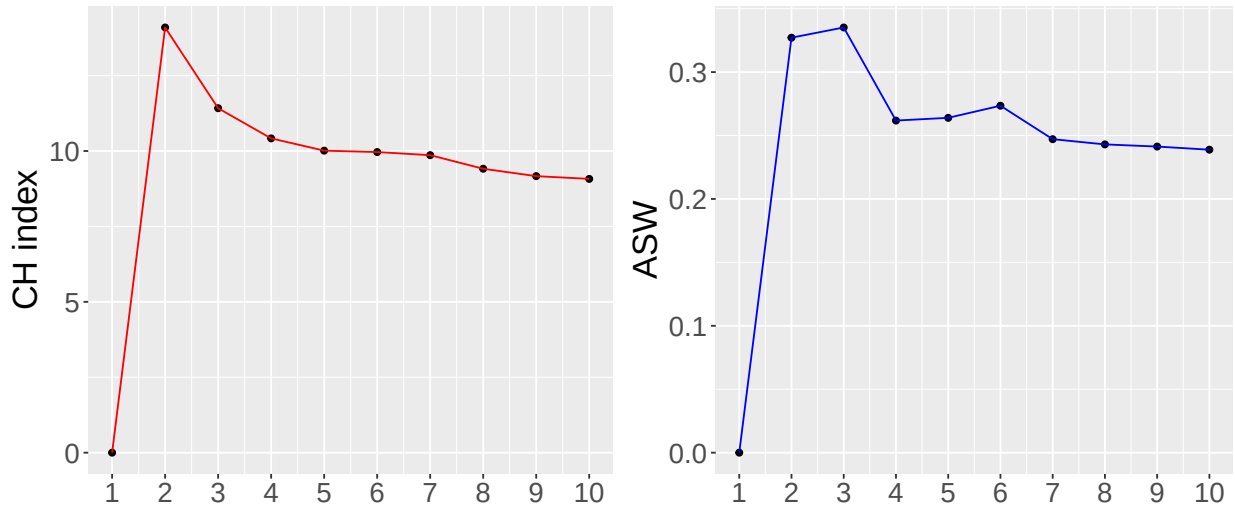
For more information about *silhouette* clustering, see http://mng.bz/Qe15

kmeansruns() for picking k (Cont.)

```
library(fpc)
kmClustering.ch <- kmeansruns(scaled_df, krange=1:10, criterion="ch")
kmClustering.ch$bestk
## [1] 2
kmClustering.asw <- kmeansruns(scaled_df, krange=1:10, criterion="asw")
kmClustering.asw$bestk
## [1] 3

# Compare the CH values for kmeans() and hclust().
print("CH index from kmeans for k=1 to 10:")
## [1] "CH index from kmeans for k=1 to 10:"
print(kmClustering.ch$crit)
##  [1]  0.000000 14.094814 11.417985 10.418801 10.011797  9.964967  9.861682
##  [8]  9.412089  9.166676  9.075569

print("CH index from hclust for k=1 to 10:")
## [1] "CH index from hclust for k=1 to 10:"
hclusting <- CH_index(scaled_df, 10, method="hclust")
print(hclusting$CH_index)
##  [1]       NaN 12.215107 10.359587  9.847229 10.011797  9.964967  9.552301
##  [8]  9.139594  8.873366  8.751161
```

Code to plot CH and ASW against k

```
library(gridExtra)
kmCritframe <- data.frame(k=1:10, ch=kmClustering.ch$crit,
                          asw=kmClustering.asw$crit)
fig1 <- ggplot(kmCritframe, aes(x=k, y=ch)) +
  geom_point() + geom_line(colour="red") +
  scale_x_continuous(breaks=1:10, labels=1:10) +
  labs(y="CH index") + theme(text=element_text(size=20))
fig2 <- ggplot(kmCritframe, aes(x=k, y=asw)) +
  geom_point() + geom_line(colour="blue") +
  scale_x_continuous(breaks=1:10, labels=1:10) +
  labs(y="ASW") + theme(text=element_text(size=20))
grid.arrange(fig1, fig2, nrow=1)
```

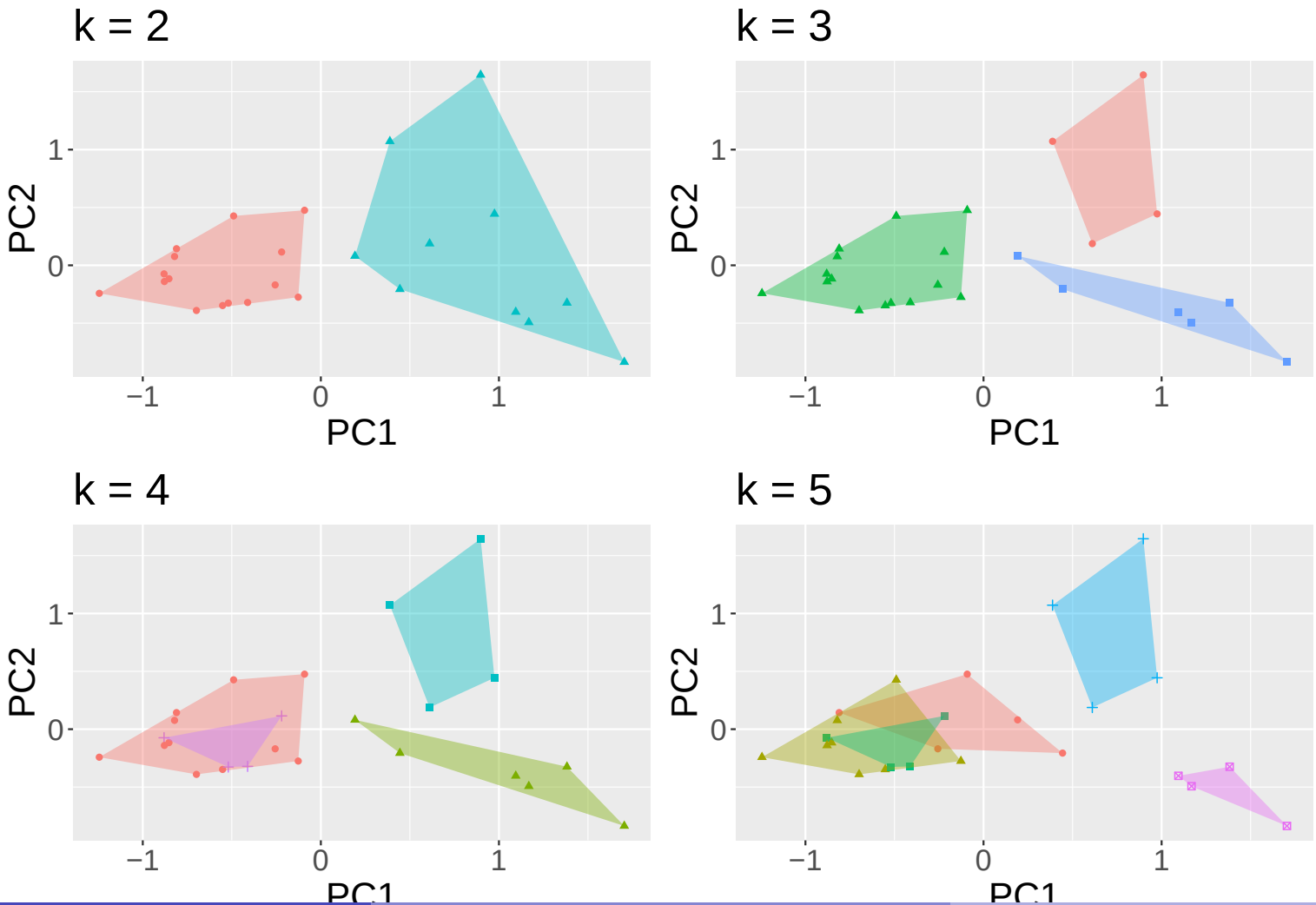


Visualisation of clusterings from kmeans

```
fig <- c()
kvalues <- seq(2,5)
for (k in kvalues) {
  groups <- kmeans(scaled_df, k, nstart=100, iter.max=100)$cluster
  kmclust.project2D <- cbind(project2D, cluster=as.factor(groups),
                             country=df$Country)
  kmclust.hull <- find_convex_hull(kmclust.project2D, groups)
  assign(paste0("fig", k),
    ggplot(kmclust.project2D, aes(x=PC1, y=PC2)) +
    geom_point(aes(shape=cluster, color=cluster)) +
    geom_polygon(data=kmclust.hull, aes(group=cluster, fill=cluster),
                 alpha=0.4, linetype=0) +
    labs(title = sprintf("k = %d", k)) +
    theme(legend.position="none", text=element_text(size=20))
    )
}
```

Visualisation of clusterings from kmeans

```
library(gridExtra)
grid.arrange(fig2, fig3, fig4, fig5, nrow=2)
```



Clustering take-aways

- The goal of clustering is to discover or draw out similarities among subsets of your data.
- In a good clustering, points in the same cluster should be more similar (nearer) to each other than they are to points in other clusters.
- When clustering, the units used in the variables matter. Different units cause different distances and potentially different clusterings.
- Ideally, you want a unit change in each coordinate to represent the same degree of change. One way to approximate this is to transform all the columns to have a mean value of 0 and a standard deviation of 1.0, for example by using the function `scale()`.

Clustering take-aways (Cont.)

- Clustering is often used for data exploration or as a precursor to supervised learning methods.
- Like visualisation, it's more iterative and interactive, and less automated than supervised methods.
- Different clustering algorithms will give different results. You should consider different approaches, with different numbers of clusters.
- There are many heuristics for estimating the best number of clusters. Again, you should consider the results from different heuristics and explore different numbers of clusters.

References

- Practical Data Science with R. By Nina Zumel and John Mount, Manning, 2014. (Chapter 9, Section 9.1, pages 312-340)