Cluster Analysis

Clustering is a branch of unsupervised learning that deals with the grouping of objects/individuals according to appropriate similarity criteria that come from our data. These groups must be

- internally homogeneous and externally heterogeneous
- few in number

Data requirements

- Low collinearity between the variables used since they should be real classification dimensions.
- Check for outliers since cluster algorithms are sensitive to extreme values.
- Standardize the data to achieve homogeneous units of measurement (avoid comparing different things).
- Data do not have to be metric/non-metric.
- Data do not have to be normally distributed/linearly related

Main steps of cluster analyses

1. Select a proximity measure (distance/similarity measure) for individual observations
2. Choose a clustering algorithm
3. Define the new distance between two clusters
4. Determine the optimal number of clusters

Proximity measures (step 1)

They describe the relationship between objects. On the basis of these relationships, the individual objects are summarized into groups. We have

- similarity measures: Pearson correlation,…
- distance measures: City block distance, (squared) Euclidean distance,…

Algorithm and new distance (steps 2 & 3)

- Single linkage (nearest neighbor): new distance is smallest individual distance
- Complete linkage (furthest neighbor): new distance is largest individual distance
- Ward: Calculation of new distance is based on a specific formula

Optimal number of clusters (step 4)

Final solution must be

- interpretable.
- the best one w.r.t. to initial research problem.
- Evaluate several solutions and choose the most suitable
- Elbow criterion for agglomeration coefficients (AC)

Example

We will use the "cars.sav" data

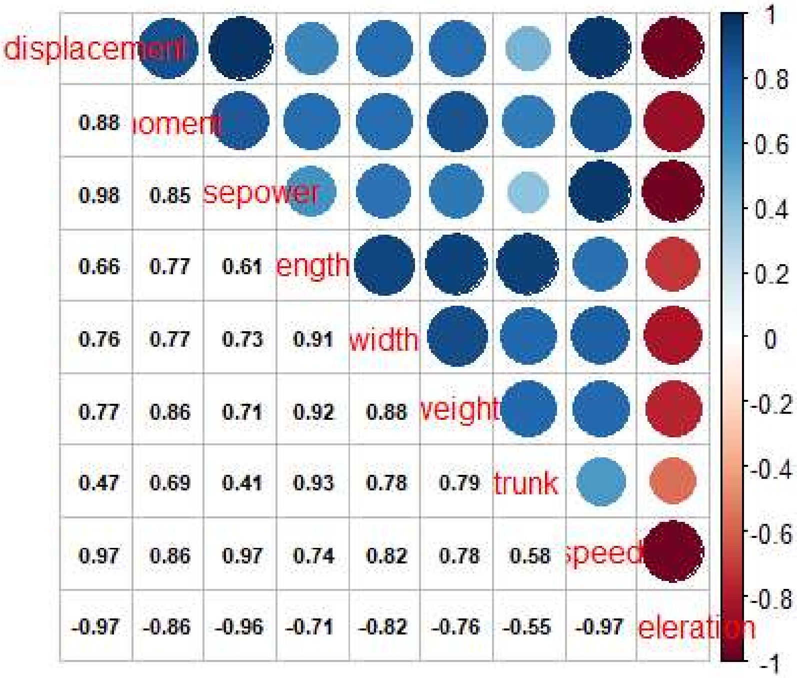
```
library(haven)
cars <- read_sav("D:/data/Empirical Research/7 Cluster
Analysis/cars.sav")
head(cars)

## # A tibble: 6 × 10
##   Name            displacement moment horsepower length width weight
trunk speed
##   <chr>                  <dbl>  <dbl>      <dbl>  <dbl> <dbl>  <dbl>
<dbl> <dbl>
## 1 Kia Picanto 1.…         1086     97         65   3535  1595    929
127   154
## 2 Suzuki Splash …          996     90         65   3715  1680   1050
178   160
## 3 Renault Clio 1…         1149    105         75   3986  1719   1155
288   167
## 4 Dacia Sandero …         1598    128         87   4020  1746   1111
320   174
## 5 Fiat Grande Pu…         1598    140         88   3986  1719   1215
288   177
## 6 Peugot 207 1.4          1360    133         88   4030  1748   1214
270   180
## # ℹ 1 more variable: acceleration <dbl>
```

He can check for pairwise collinearity among the variables

```
library(corrplot)

corrplot.mixed (cor(cars[,2:10]), lower.col='black', number.cex=.7)
```



As we can see there is indeed correlation among various variables, which means that ideally we should discard some of those. Additionally we could also check for outliers(using boxplots for instance). However, we are gonna keep the complete data set for our analysis. On the other hand, it is vital to standardize the data before proceeding

```
cardat = scale(cars[,2:10])
row.names(cardat) <- cars$Name
```

As a first step let us select the "euclidean distance" as proximity measure for our data. We calculate the distance of each individual pair. This can be done either manually for each pair as the example below

```
cars[,2:10] = scale(cars[,2:10])
edman = ((cars[1,2]-cars[2,2])^2+(cars[1,3]-cars[2,3])^2+(cars[1,4]-
cars[2,4])^2+
        (cars[1,5]-cars[2,5])^2+(cars[1,6]-cars[2,6])^2+(cars[1,7]-
cars[2,7])^2+
        (cars[1,8]-cars[2,8])^2+(cars[1,9]-cars[2,9])^2+(cars[1,10]-
cars[2,10])^2)^0.5
edman

##   displacement
## 1     1.415792
```

or (more preferably) automatically for all pairs by using the appropriate library in

```
eucdist = dist(cardat, method = "euclidean") #city block = manhattan
eucdist

##                         Kia Picanto 1.1 Start Suzuki Splash 1.0
Renault Clio 1.2
## Suzuki Splash 1.0                   1.4157920
## Renault Clio 1.2                    2.5470040          1.2901173
## Dacia Sandero 1.6                   3.1749981          2.0668398
1.0189627
## Fiat Grande Punto 1.4               2.9522999          1.8969854
0.9134813
## Peugot 207 1.4                      3.0879239          1.8497119
0.7690821
## Renault Clio 1.6                    2.6108615          1.7138905
0.9619760
## Porsche Cayman                      7.7791344          7.0958946
6.2359422
## Nissan 350Z                         7.8134078          7.0971321
6.2949688
## Mercedes C 200 CDI                  5.8809554          4.9145043
3.7786405
## VWPassat Variant 2.0                6.7308249          5.7031897
4.5142447
## Skoda Octavia 2.0                   6.0413659          5.1404868
4.0034194
## Mercedes E 280                      7.8973044          6.9912346
5.8821730
## Audi A6 2.4                         7.1523072          6.0845016
4.8781798
## BMW 525i                            7.4364202          6.4364657
5.2843718
##                         Dacia Sandero 1.6 Fiat Grande Punto 1.4 Peugot
207 1.4
## Suzuki Splash 1.0
## Renault Clio 1.2
## Dacia Sandero 1.6
## Fiat Grande Punto 1.4           0.6457943
## Peugot 207 1.4                  0.7488732             0.5965032
## Renault Clio 1.6                0.9379264             0.6833489
1.0210708
## Porsche Cayman                  5.4354698             5.4421008
5.6019002
## Nissan 350Z                     5.5852066             5.4775851
5.6124943
## Mercedes C 200 CDI              3.2995636             3.1596306
3.2306663
## VWPassat Variant 2.0            4.0055701             3.9964228
3.9940976
## Skoda Octavia 2.0               3.4163050             3.4098296
3.5184166
```

```
## Mercedes E 280                 5.1950803            5.1680707
5.2571568
## Audi A6 2.4                    4.2180050            4.3021384
4.2777889
## BMW 525i                       4.5867724            4.6346324
4.6543360
##                        Renault Clio 1.6 Porsche Cayman Nissan 350Z
## Suzuki Splash 1.0
## Renault Clio 1.2
## Dacia Sandero 1.6
## Fiat Grande Punto 1.4
## Peugot 207 1.4
## Renault Clio 1.6
## Porsche Cayman                5.6547339
## Nissan 350Z                   5.7699108      1.7497653
## Mercedes C 200 CDI            3.5516343      3.8714887   3.8760947
## VWPassat Variant 2.0          4.3851590      4.1115952   4.2429786
## Skoda Octavia 2.0             3.7092358      3.7150903   4.1088019
## Mercedes E 280                5.4878224      2.3995923   2.6594090
## Audi A6 2.4                   4.6499825      3.3444373   3.6575656
## BMW 525i                      4.9422793      2.6431069   3.0062098
##                        Mercedes C 200 CDI VWPassat Variant 2.0 Skoda
Octavia 2.0
## Suzuki Splash 1.0
## Renault Clio 1.2
## Dacia Sandero 1.6
## Fiat Grande Punto 1.4
## Peugot 207 1.4
## Renault Clio 1.6
## Porsche Cayman
## Nissan 350Z
## Mercedes C 200 CDI
## VWPassat Variant 2.0            1.2023888
## Skoda Octavia 2.0               1.1735730            1.1507777
## Mercedes E 280                  2.4868732            2.4274689
2.6068039
## Audi A6 2.4                     1.8677239            1.6464649
2.0131658
## BMW 525i                        2.2260448            2.1371294
2.3108734
##                        Mercedes E 280 Audi A6 2.4
## Suzuki Splash 1.0
## Renault Clio 1.2
## Dacia Sandero 1.6
## Fiat Grande Punto 1.4
## Peugot 207 1.4
## Renault Clio 1.6
## Porsche Cayman
## Nissan 350Z
## Mercedes C 200 CDI
```

```
## VWPassat Variant 2.0
## Skoda Octavia 2.0
## Mercedes E 280
## Audi A6 2.4                 1.6055452
## BMW 525i                    1.0080207   0.8669581
```

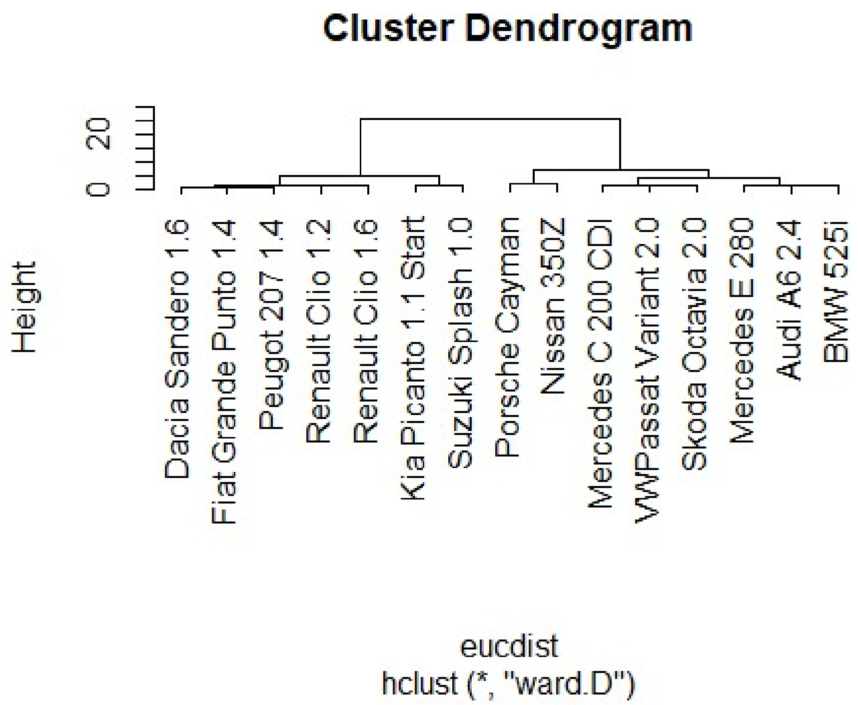
For squared euclidean and other distance measures: use distance() from philentropy

Now we perform clustering by defining the specific algorithm and the new distance rule for the procedure. Foe example

```
#single linkage=single, complete linkage=complete
clus = hclust(eucdist, method="ward.D")
clus

##
## Call:
## hclust(d = eucdist, method = "ward.D")
##
## Cluster method   : ward.D
## Distance         : euclidean
## Number of objects: 15

plot(clus)
```



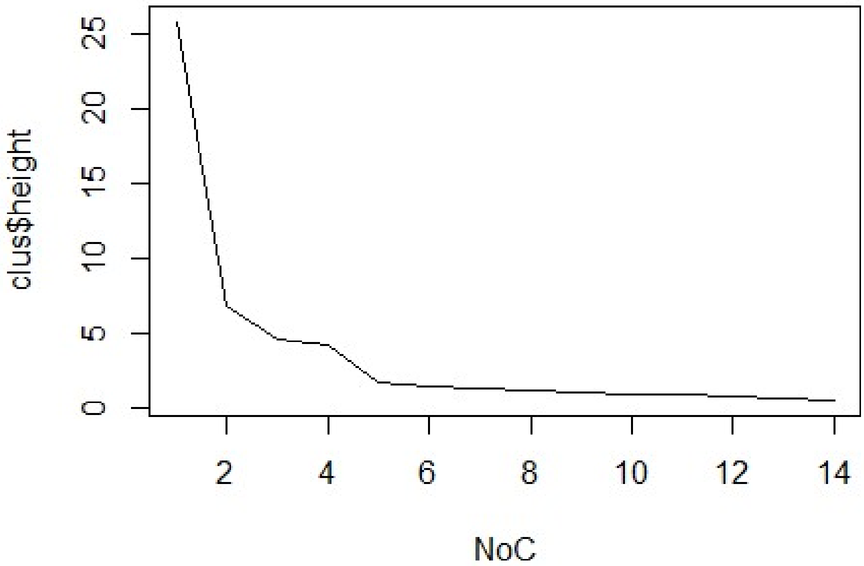

```
data.frame(clus[2:1])
```

```
##         height merge.1 merge.2
## 1   0.5965032      -5      -6
## 2   0.7309439      -4       1
## 3   0.8669581     -14     -15
## 4   0.9619760      -3      -7
## 5   1.0293845       2       4
## 6   1.1507777     -11     -12
## 7   1.2003820     -10       6
## 8   1.4157920      -1      -2
## 9   1.4533912     -13       3
## 10  1.7497653      -8      -9
## 11  4.2384283       7       9
## 12  4.6663842       5       8
## 13  6.8692843      10      11
## 14 25.8322239      12      13
```

We observe that the result varies from the one extreme case of only one cluster(where all the 15 individuals belong to a single group) to the other extreme case of 15 clusters(where each individual belongs to each own group). Both case are quite useless and not informative.

The optimal number of clusters can be determined observing the change-rate of *height* as we move alongs stage with ascending clusters

```
NoC = 14:1
plot(NoC,clus$height, type="l")
```



As we can see the the most radical change of $height$ occurs as soon as we reach the stage with 2 clusters. Thus, according to the $elbow - rule$ we must choose 2 clusters for our analysis

```
plot(clus)
rect.hclust(clus,k=2,border="red")
```