Data Science with R
Cluster Analysis

Graham.Williams@togaware.com

22nd June 2014

Visit http://onepager.togaware.com/ for more OnePageR's.

We focus on the unsupervised method of cluster analysis in this chapter. Cluster analysis is a topic that has been much studied by statisticians for decades and widely used in data mining.

The required packages for this module include:

```
library(rattle)       # The weather dataset and normVarNames().
library(randomForest) # Impute missing values using na.roughfix().
library(ggplot2)      # Visualise the data through plots.
library(animation)    # Demonstrate kmeans.
library(reshape2)     # Reshape data for plotting.
library(fpc)          # Tuning clusterng with kmeansruns() and clusterboot().
library(clusterCrit)  # Clustering criteria.
library(wskm)         # Weighted subspace clustering.
library(amap)         # hclusterpar
library(cba)          # Dendrogram plot
library(dendroextras) # To colour clusters
library(kohonen)      # Self organising maps.
```

As we work through this chapter, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the *help=* option of `library()`:

```
library(help=rattle)
```

This chapter is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

1 Load Weather Dataset for Modelling

We use the **weather** dataset from rattle (Williams, 2014) and normalise the variable names. Missing values are imputed using `na.roughfix()` from randomForest (Breiman *et al.*, 2012), particularly because `kmeans()` does not handle missing values itself. Here we set up the dataset for modelling. Notice in particular we identify the numeric input variables (*numi* is an integer vector containing the column index for the numeric variables and *numc* is a character vector containing the column names). Many clustering algorithms only handle numeric variables.

```
# Required packages
library(rattle)         # Load weather dataset. Normalise names normVarNames().
library(randomForest)   # Impute missing using na.roughfix().

# Identify the dataset.
dsname      <- "weather"
ds          <- get(dsname)
names(ds)   <- normVarNames(names(ds))
vars        <- names(ds)
target      <- "rain_tomorrow"
risk        <- "risk_mm"
id          <- c("date", "location")

# Ignore the IDs and the risk variable.
ignore      <- union(id, if (exists("risk")) risk)

# Ignore variables which are completely missing.
mvc         <- sapply(ds[vars], function(x) sum(is.na(x))) # Missing value count.
mvn         <- names(ds)[(which(mvc == nrow(ds)))]         # Missing var names.
ignore      <- union(ignore, mvn)

# Initialise the variables
vars        <- setdiff(vars, ignore)

# Variable roles.
inputc      <- setdiff(vars, target)
inputi      <- sapply(inputc, function(x) which(x == names(ds)), USE.NAMES=FALSE)
numi        <- intersect(inputi, which(sapply(ds, is.numeric)))
numc        <- names(ds)[numi]
cati        <- intersect(inputi, which(sapply(ds, is.factor)))
catc        <- names(ds)[cati]

# Impute missing values, but do this wisely - understand why missing.
if (sum(is.na(ds[vars]))) ds[vars] <- na.roughfix(ds[vars])

# Number of observations.
nobs        <- nrow(ds)
```

2 Introducing Cluster Analysis

The aim of cluster analysis is to identify groups of observations so that within a group the observations are most similar to each other, whilst between groups the observations are most dissimilar to each other. Cluster analysis is essentially an unsupervised method.

Our human society has been "clustering" for a long time to help us understand the environment we live in. We have clustered the animal and plant kingdoms into a hierarchy of similarities. We cluster chemical structures. Day-by-day we see grocery items clustered into similar groups. We cluster student populations into similar groups of students from similar backgrounds or studying similar combinations of subjects.

The concept of similarity is often captured through the measurement of distance. Thus we often describe cluster analysis as identifying groups of observations so that the distance between the observations within a group is minimised and between the groups the distance is maximised. Thus a distance measure is fundamental to calculating clusters.

There are some caveats to performing automated cluster analysis using distance measures. We often observe, particularly with large datasets, that a number of interesting clusters will be generated, and then one or two clusters will account for the majority of the observations. It is as if these larger clusters simply lump together those observations that don't fit elsewhere.

3 Distance Calculation: Euclidean Distance

Suppose we pick the first two observations from our dataset and the first 5 numeric variables:

```
ds[1:2, numi[1:5]]
```

```
##   min_temp max_temp rainfall evaporation sunshine
## 1        8     24.3      0.0         3.4      6.3
## 2       14     26.9      3.6         4.4      9.7
```

```
x <- ds[1, numi[1:5]]
y <- ds[2, numi[1:5]]
```

Then $x - y$ is simply:

```
x-y
```

```
##   min_temp max_temp rainfall evaporation sunshine
## 1       -6     -2.6     -3.6          -1     -3.4
```

Then the square of each difference is:

```
sapply(x-y, '^', 2)
```

```
##    min_temp    max_temp    rainfall evaporation    sunshine
##       36.00        6.76       12.96        1.00       11.56
```

The sum of the squares of the differences:

```
sum(sapply(x-y, '^', 2))
```

```
## [1] 68.28
```

Finally the square root of the sum of the squares of the differences (also known as the Euclidean distance) is:

```
sqrt(sum(sapply(x-y, '^', 2)))
```

```
## [1] 8.263
```

Of course we don't need to calculate this so manually ourselves. R provides `dist()` to calculate the distance (Euclidean distance by default):

```
dist(ds[1:2, numi[1:5]])
```

```
##       1
## 2 8.263
```

We can also calculate the Manhattan distance:

```
sum(abs(x-y))
```

```
## [1] 16.6
```

```
dist(ds[1:2, numi[1:5]], method="manhattan")
```

```
##      1
## 2 16.6
```

4 Minkowski Distance

```
dist(ds[1:2, numi[1:5]], method="minkowski", p=1)
```

```
##      1
## 2 16.6
```

```
dist(ds[1:2, numi[1:5]], method="minkowski", p=2)
```

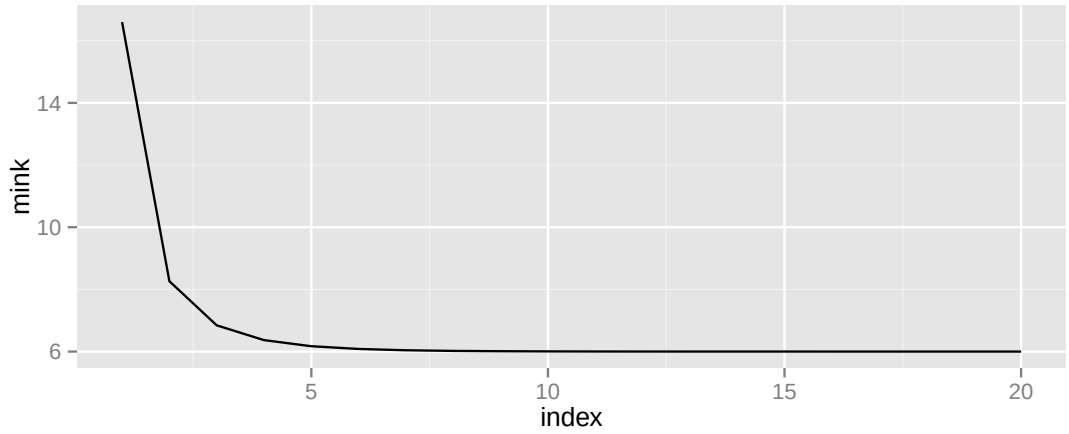
```
##       1
## 2 8.263
```

```
dist(ds[1:2, numi[1:5]], method="minkowski", p=3)
```

```
##       1
## 2 6.844
```

```
dist(ds[1:2, numi[1:5]], method="minkowski", p=4)
```

```
##       1
## 2 6.368
```



5 General Distance

```
dist(ds[1:5, numi[1:5]])
```

```
##        1      2      3      4
## 2  8.263
## 3  7.812  7.434
## 4 41.375 38.067 37.531
....
```

```
daisy(ds[1:5, numi[1:5]])
```

```
## Dissimilarities :
##        1      2      3      4
## 2  8.263
## 3  7.812  7.434
....
```

```
daisy(ds[1:5, cati])
```

```
## Dissimilarities :
##        1      2      3      4
## 2 0.6538
## 3 0.6923 0.5385
....
```

6 K-Means Basics: Iterative Cluster Search

The k-means algorithm is a traditional and widely used clustering algorithm.

The algorithm begins by specifying the number of clusters we are interested in—this is the k. Each of the k clusters is identified as the vector of the average (i.e., the mean) value of each of the variables for observations within a cluster. A random clustering is first constructed, the k means calculated, and then using the distance measure we gravitate each observation to its nearest mean. The means are then recalculated and the points re-gravitate. And so on until there is no change to the means.

7 K-Means: Using kmeans()

Here is our first attempt to cluster our dataset.

```
model <- m.km <- kmeans(ds, 10)

## Warning:  NAs introduced by coercion

## Error:  NA/NaN/Inf in foreign function call (arg 1)
```

The error is because there are non-numeric variables that we are attempting to cluster on.

```
set.seed(42)
model <- m.km <- kmeans(ds[numi], 10)
```

So that appears to have succeeded to build 10 clusters. The sizes of the clusters can readily be listed:

```
model$size

##  [1] 29 47 24 55 21 33 35 50 41 31
```

The cluster centers (i.e., the *means*) can also be listed:

```
model$centers

##    min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## 1    5.8448    20.38  0.75862       6.000   10.524           52.66
## 2   13.0340    31.42  0.09362       7.677   10.849           43.32
## 3   13.9833    21.02  7.14167       4.892    2.917           39.54
....
```

The component `m.km$cluster` reports which of the 10 clusters each of the original observations belongs:

```
head(model$cluster)

## [1]  4  8  6  6  6 10
```

```
model$iter

## [1] 6

model$ifault

## [1] 0
```

8 Scaling Datasets

We noted earlier that a unit of distance is different for differently measure variables. For example, one year of difference in age seems like it should be a larger difference than $1 difference in our income. A common approach is to rescale our data by subtracting the mean and dividing by the standard deviation. This is often referred to as a z-score. The result is that the mean for all variables is 0 and a unit of difference is one standard deviation.

The R function `scale()` can perform this transformation on our numeric data. We can see the effect in the following:

```
summary(ds[numi[1:5]])
```

```
##     min_temp        max_temp       rainfall      evaporation
##  Min.   :-5.30   Min.   : 7.6   Min.   : 0.00   Min.   : 0.20
##  1st Qu.: 2.30   1st Qu.:15.0   1st Qu.: 0.00   1st Qu.: 2.20
##  Median : 7.45   Median :19.6   Median : 0.00   Median : 4.20
....
```

```
summary(scale(ds[numi[1:5]]))
```

```
##     min_temp          max_temp         rainfall       evaporation
##  Min.   :-2.0853   Min.   :-1.936   Min.   :-0.338   Min.   :-1.619
##  1st Qu.:-0.8241   1st Qu.:-0.826   1st Qu.:-0.338   1st Qu.:-0.870
##  Median : 0.0306   Median :-0.135   Median :-0.338   Median :-0.121
....
```

The `scale()` function also provides some extra information, recording the actual original means and the standard deviations:

```
dsc <- scale(ds[numi[1:5]])
attr(dsc, "scaled:center")
```

```
##    min_temp    max_temp    rainfall evaporation    sunshine
##       7.266      20.550       1.428       4.522       7.915
```

```
attr(dsc, "scaled:scale")
```

```
##    min_temp    max_temp    rainfall evaporation    sunshine
##       6.026       6.691       4.226       2.669       3.468
```

Compare that information with the output from `mean()` and `sd()`:

```
sapply(ds[numi[1:5]], mean)
```

```
##    min_temp    max_temp    rainfall evaporation    sunshine
##       7.266      20.550       1.428       4.522       7.915
```

```
sapply(ds[numi[1:5]], sd)
```

```
##    min_temp    max_temp    rainfall evaporation    sunshine
##       6.026       6.691       4.226       2.669       3.468
```

9 K-Means: Scaled Dataset

```
set.seed(42)
model <- m.kms <- kmeans(scale(ds[numi]), 10)
model$size
```

```
##  [1] 34 54 15 70 24 32 30 44 43 20
```

```
model$centers
```

```
##    min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## 1    1.0786   1.6740 -0.31018     1.43079   1.0397          0.6088
## 2    0.5325   0.9939 -0.24074     0.56206   0.8068         -0.2149
## 3    0.8808  -0.2307  3.77323     0.01928  -0.7599          0.4886
....
```

```
model$totss
```

```
## [1] 5840
```

```
model$withinss
```

```
##  [1] 249.4 272.4 211.2 328.0 149.2 287.7 156.8 366.2 262.0 137.0
```

```
model$tot.withinss
```

```
## [1] 2420
```

```
model$betweenss
```

```
## [1] 3420
```

```
model$iter
```

```
## [1] 8
```

```
model$ifault
```

```
## [1] 0
```

10 Animate Cluster Building

Using `kmeans.ani()` from animation (Xie, 2013), we can produce an animation that illustrates the kmeans algorithm.

```
library(animation)
```

We generate some random data for two variables over 100 observations.

```
cent <- 1.5 * c(1, 1, -1, -1, 1, -1, 1, -1)
x    <- NULL
for (i in 1:8) x <- c(x, rnorm(25, mean=cent[i]))
x  <- matrix(x, ncol=2)
colnames(x) <- c("X1", "X2")

dim(x)
```

```
## [1] 100   2
```

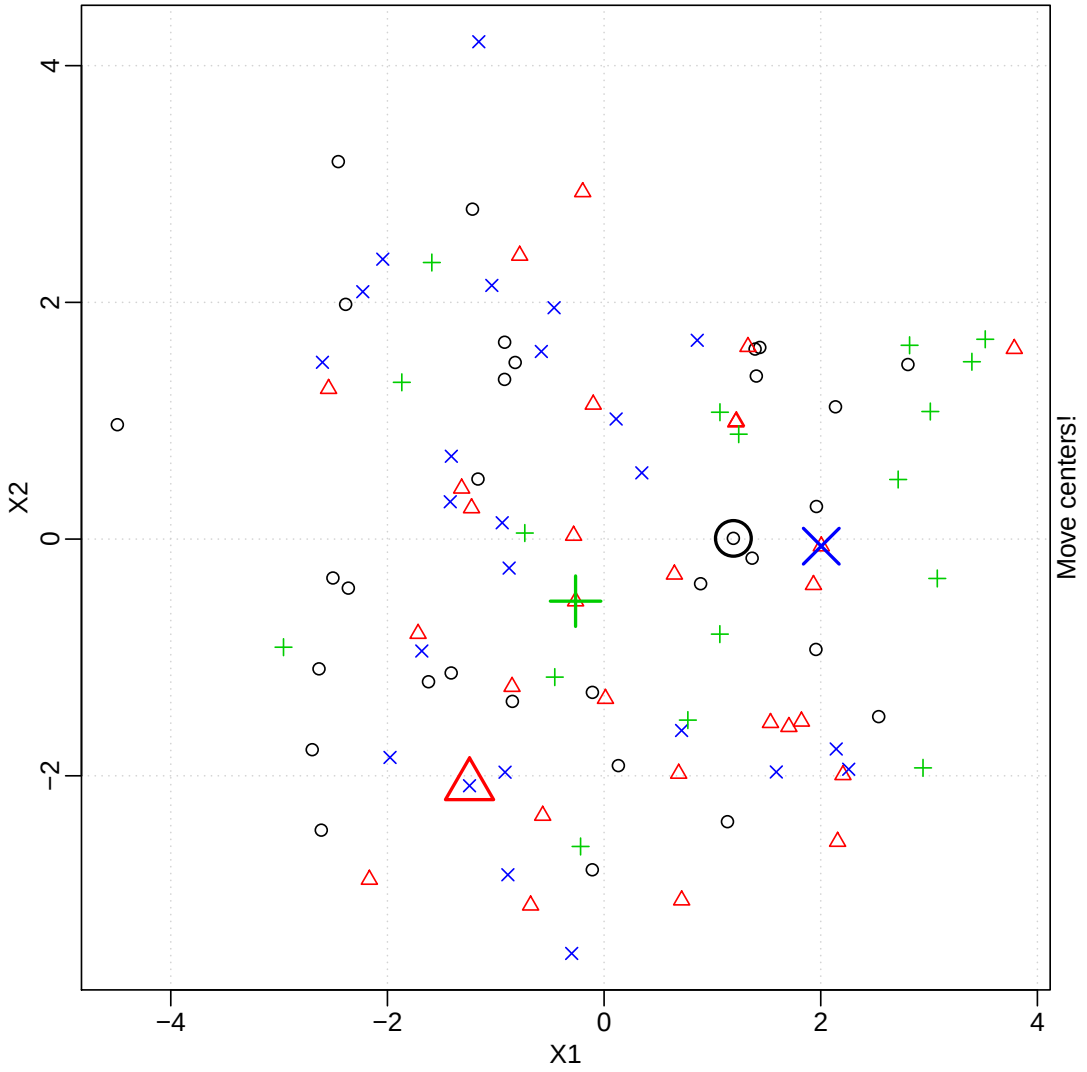
```
head(x)
```

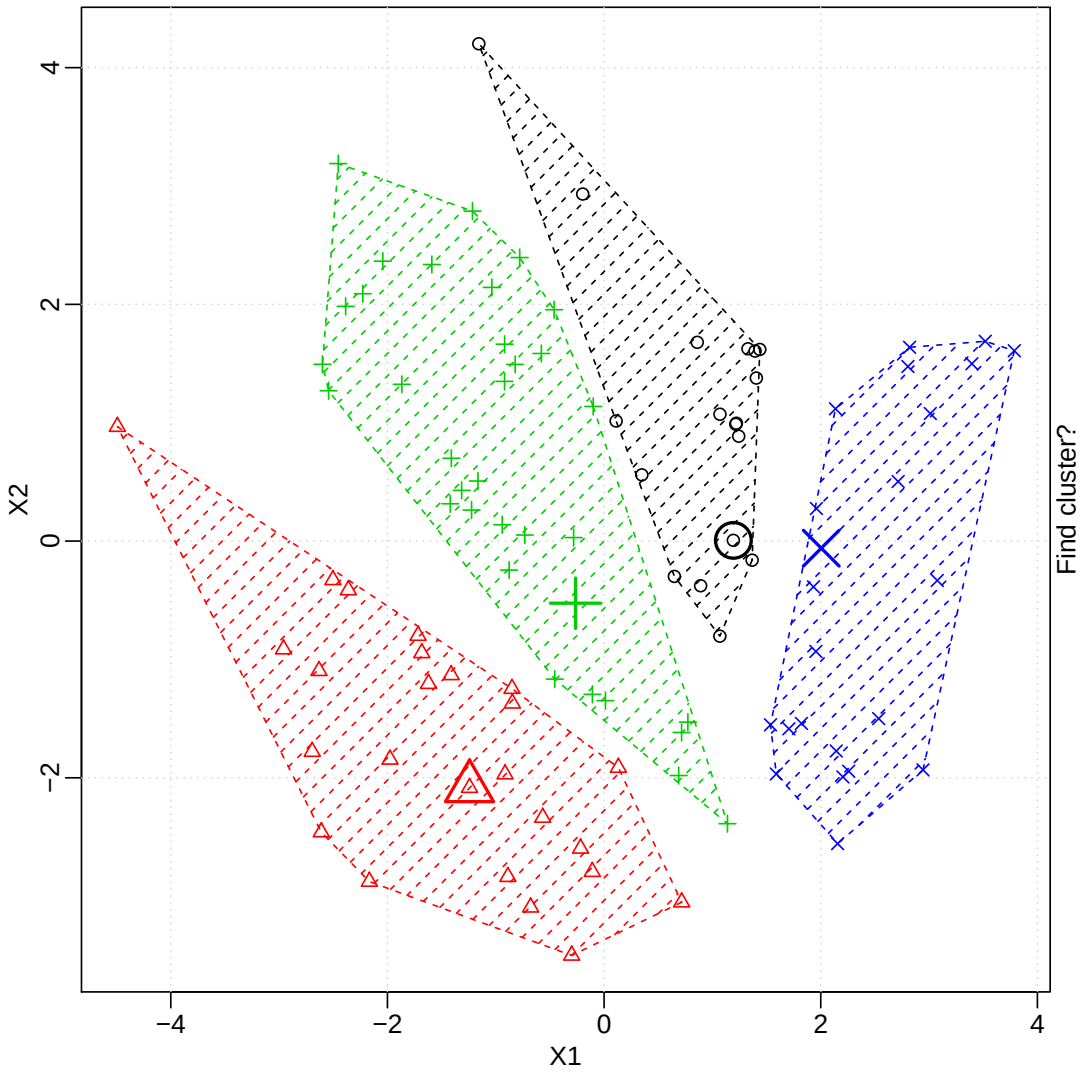
```
##          X1    X2
## [1,] 1.394 1.606
## [2,] 3.012 1.078
## [3,] 1.405 1.378
....
```

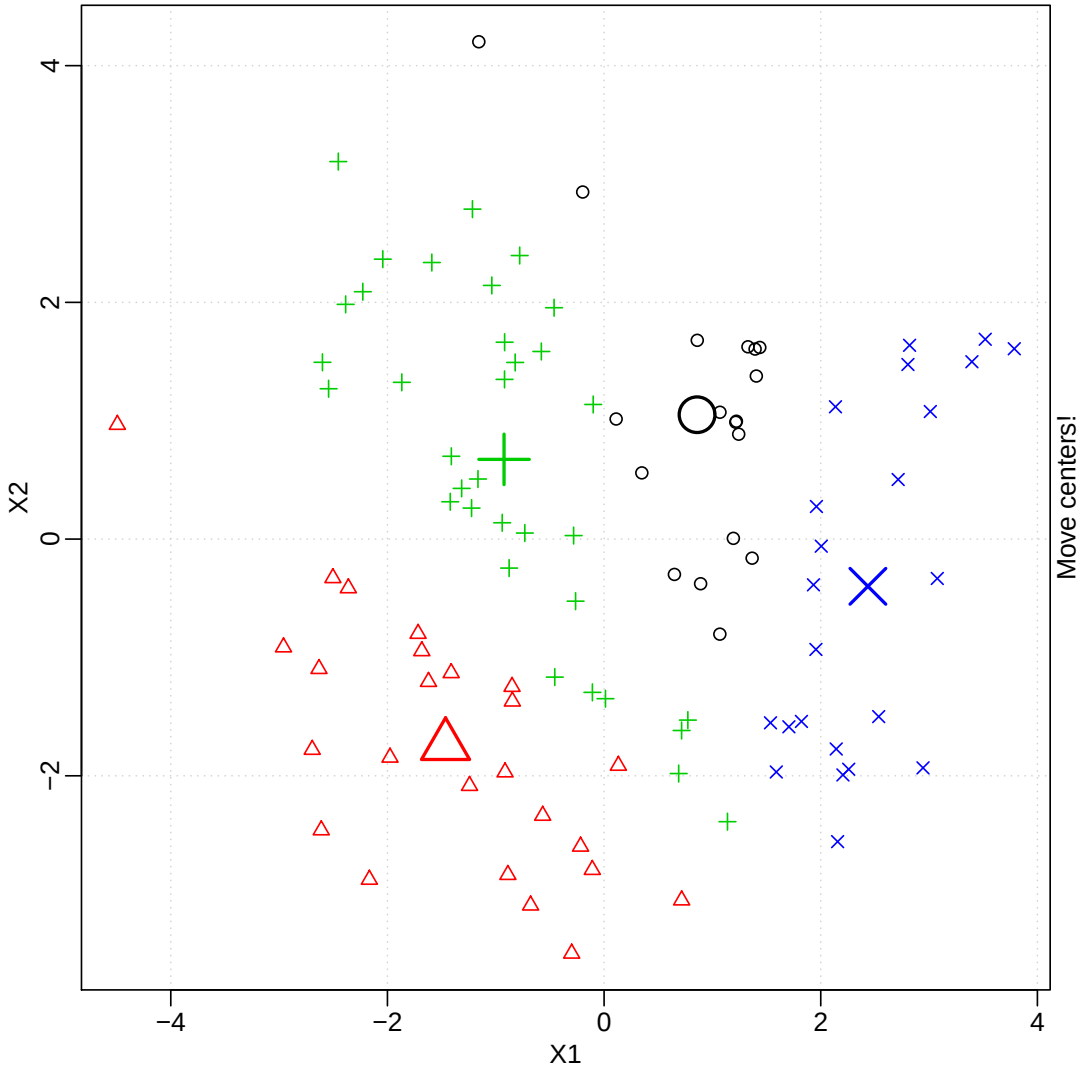
The series of plots over the following pages show the convergence of the kmeans algorithm to identify 4 clusters.

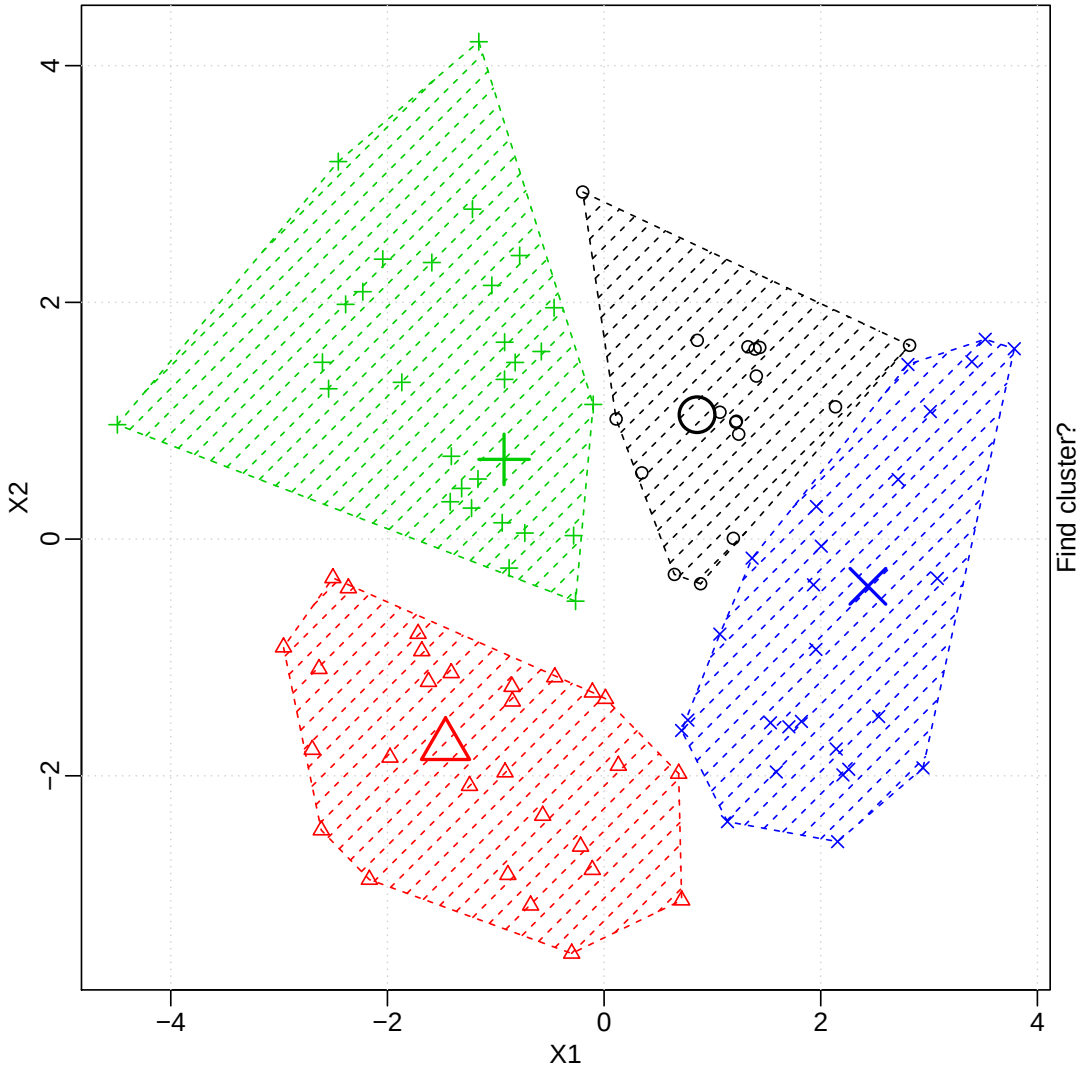
```
par(mar=c(3, 3, 1, 1.5), mgp=c(1.5, 0.5, 0), bg="white")
kmeans.ani(x, centers=4, pch=1:4, col=1:4)
```

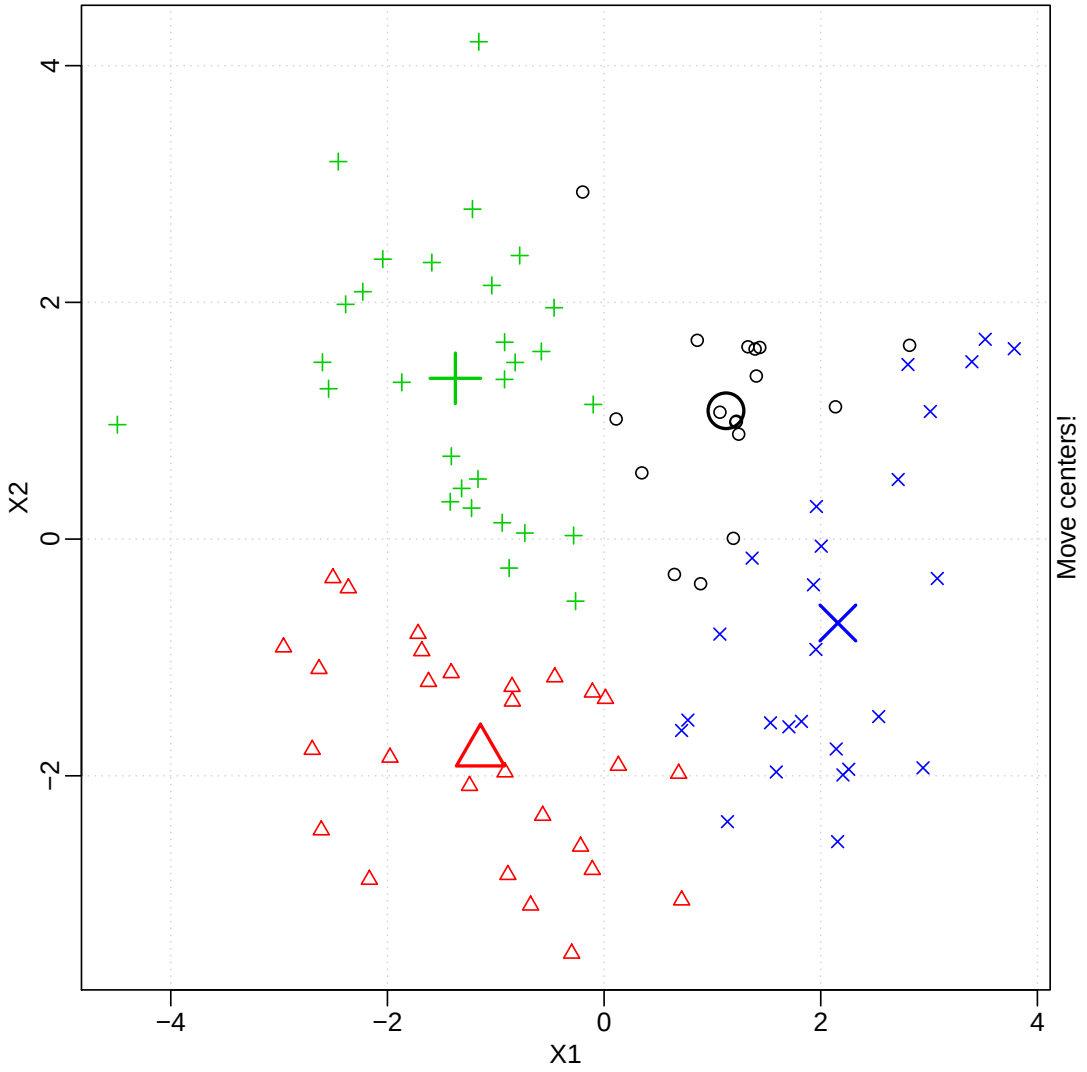
The first plot, on the next page, shows a random allocation of points to one of the four clusters, together with 4 random means. The points are then mapped to their closest means, to define the four clusters we see in the second plot. The means are then recalculated for each of the clusters, as seen in the third plot. The following plots then iterate between showing the means nearest each of the points, then re-calculating the means. Eventually, the means do not change location, and the algorithm converges.

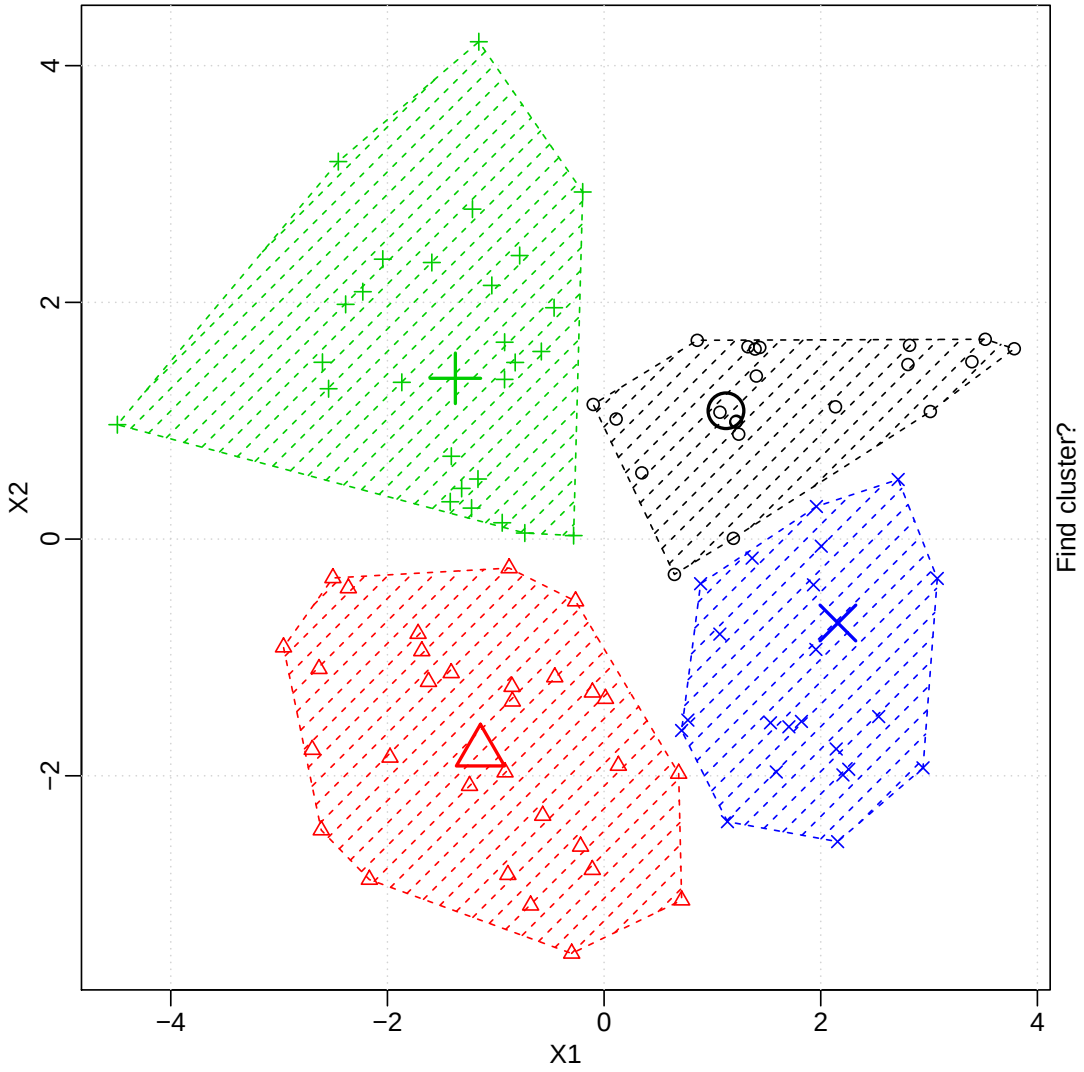

X2
X1
Move centers!
4
2
0
−2
−4
−2
0
2
4

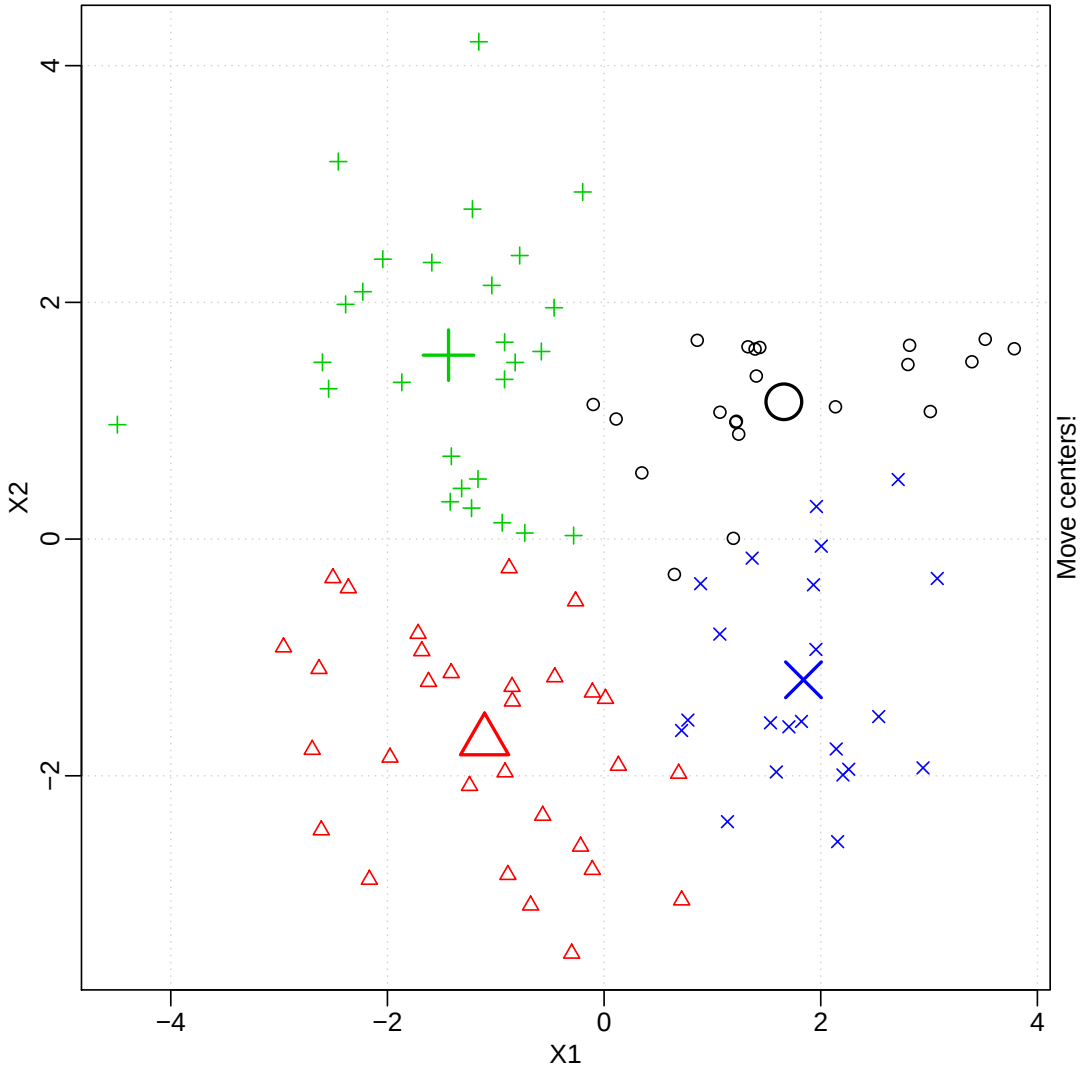

X2
X1
Find cluster?
4
2
0
−2
−4
−2
0
2
4

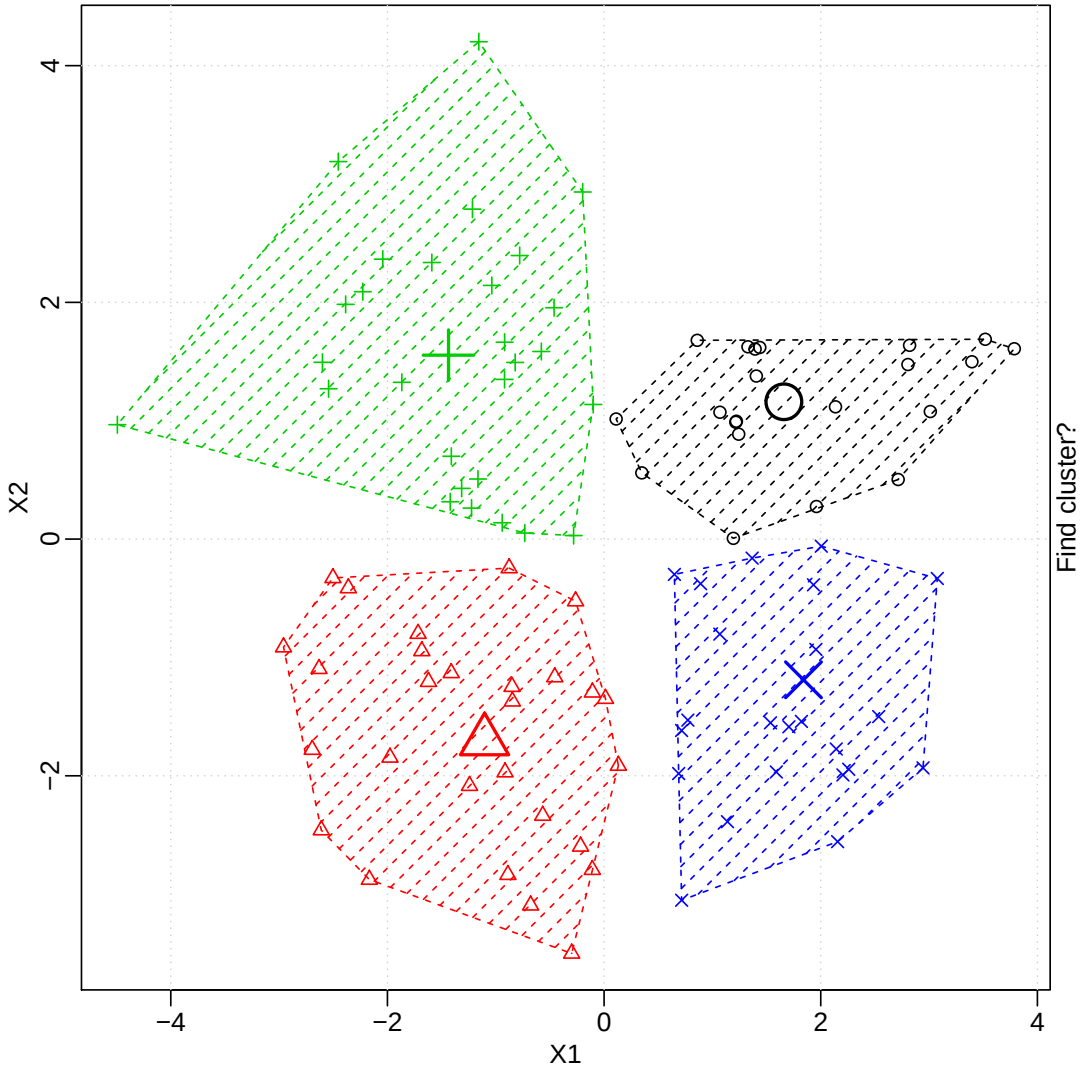

X2
X1
Move centers!
4
2
0
−2
−4
−2
0
2
4

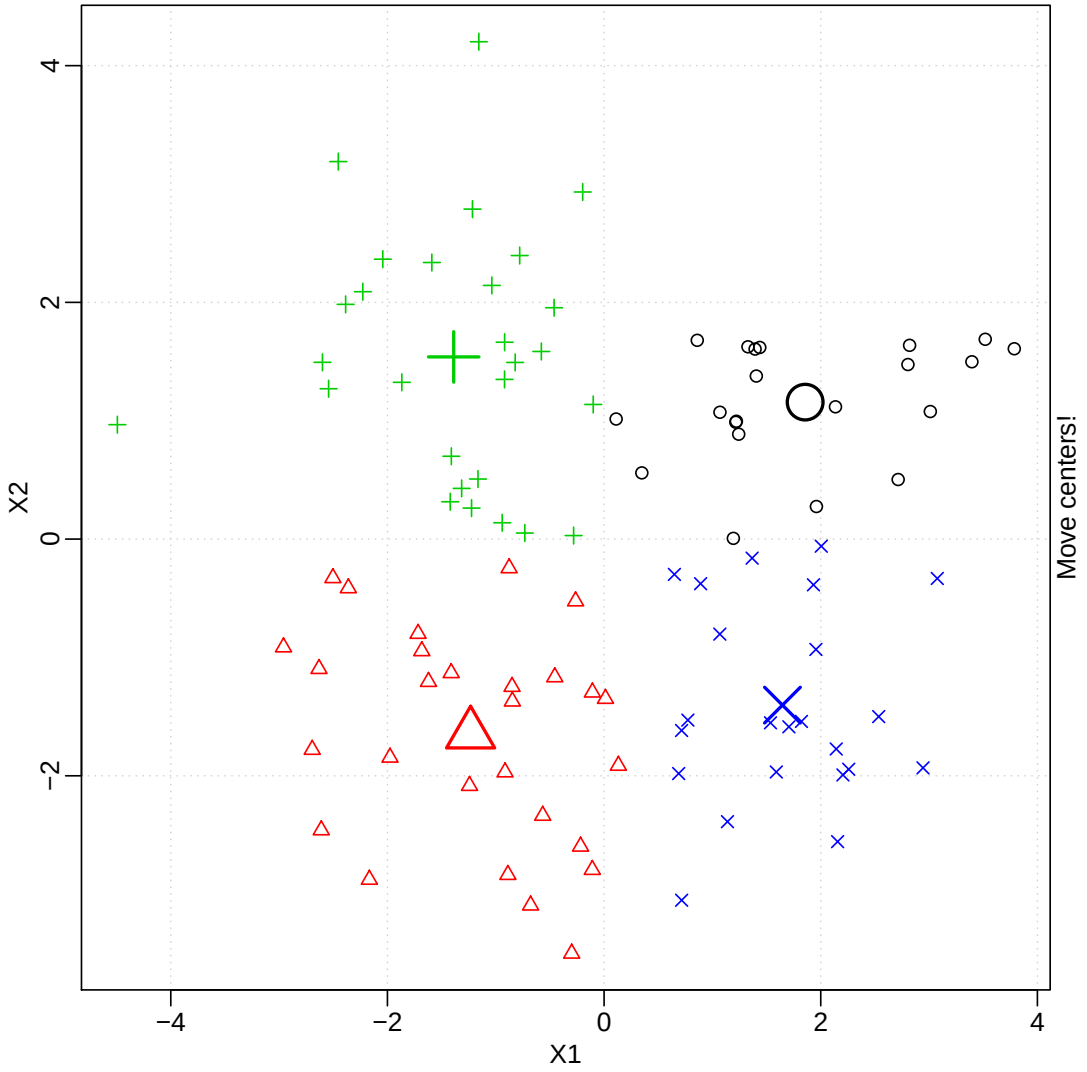

X2
4
2
0
−2
−4
−2
0
2
4
X1
Find cluster?

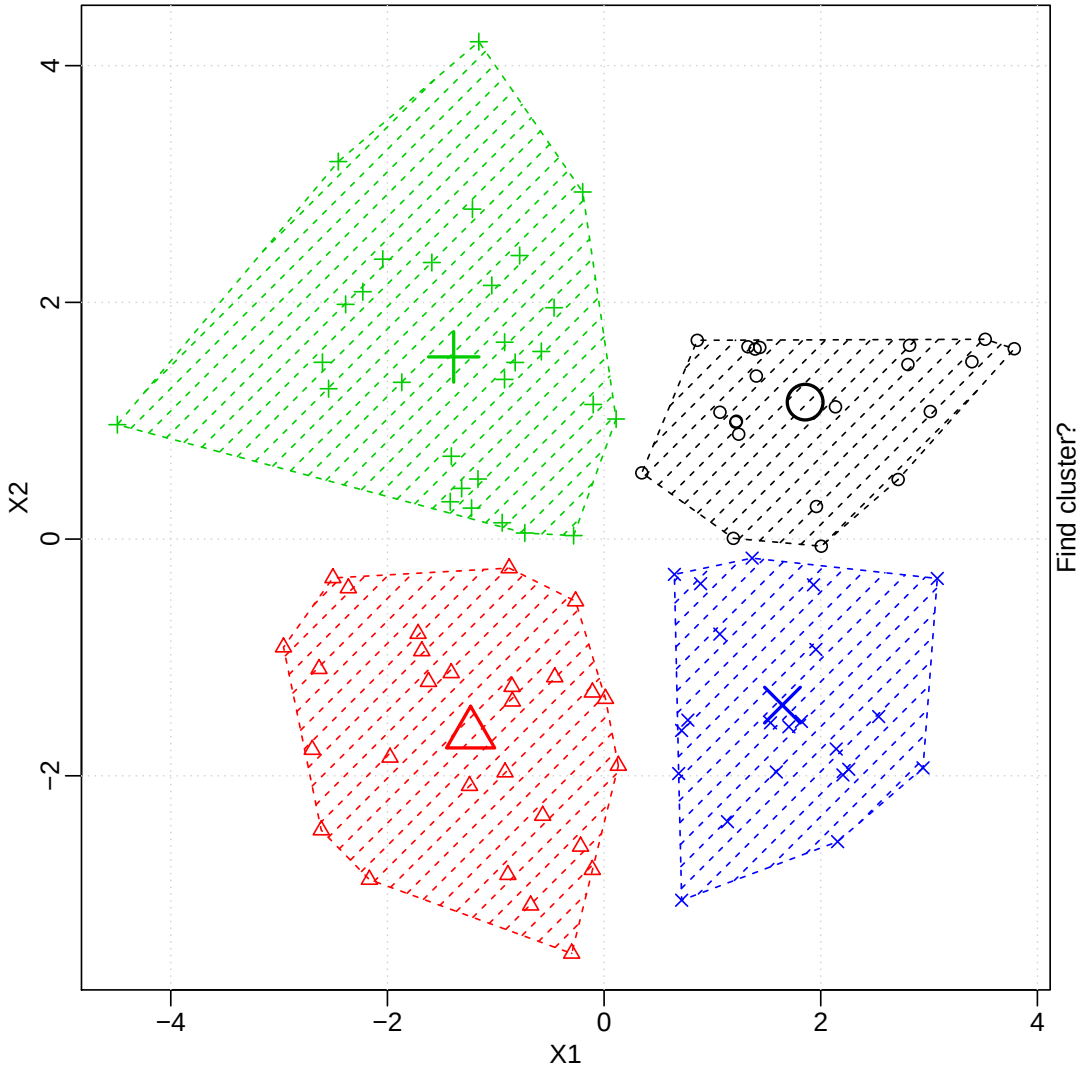

X2
X1
Move centers!
4
2
0
−2
−4
−2
0
2
4

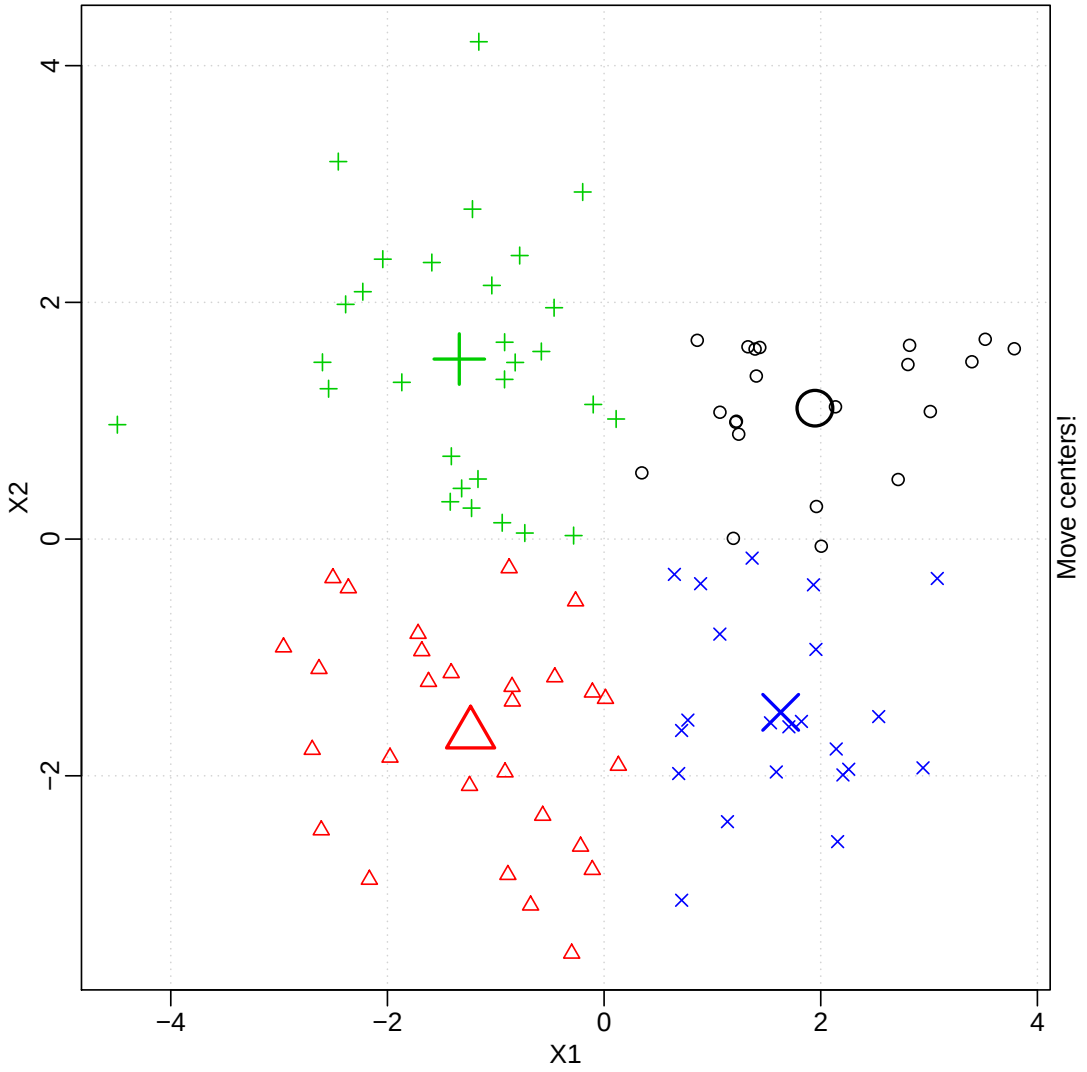

X2
X1
Find cluster?
4
2
0
−2
−4
−2
0
2
4

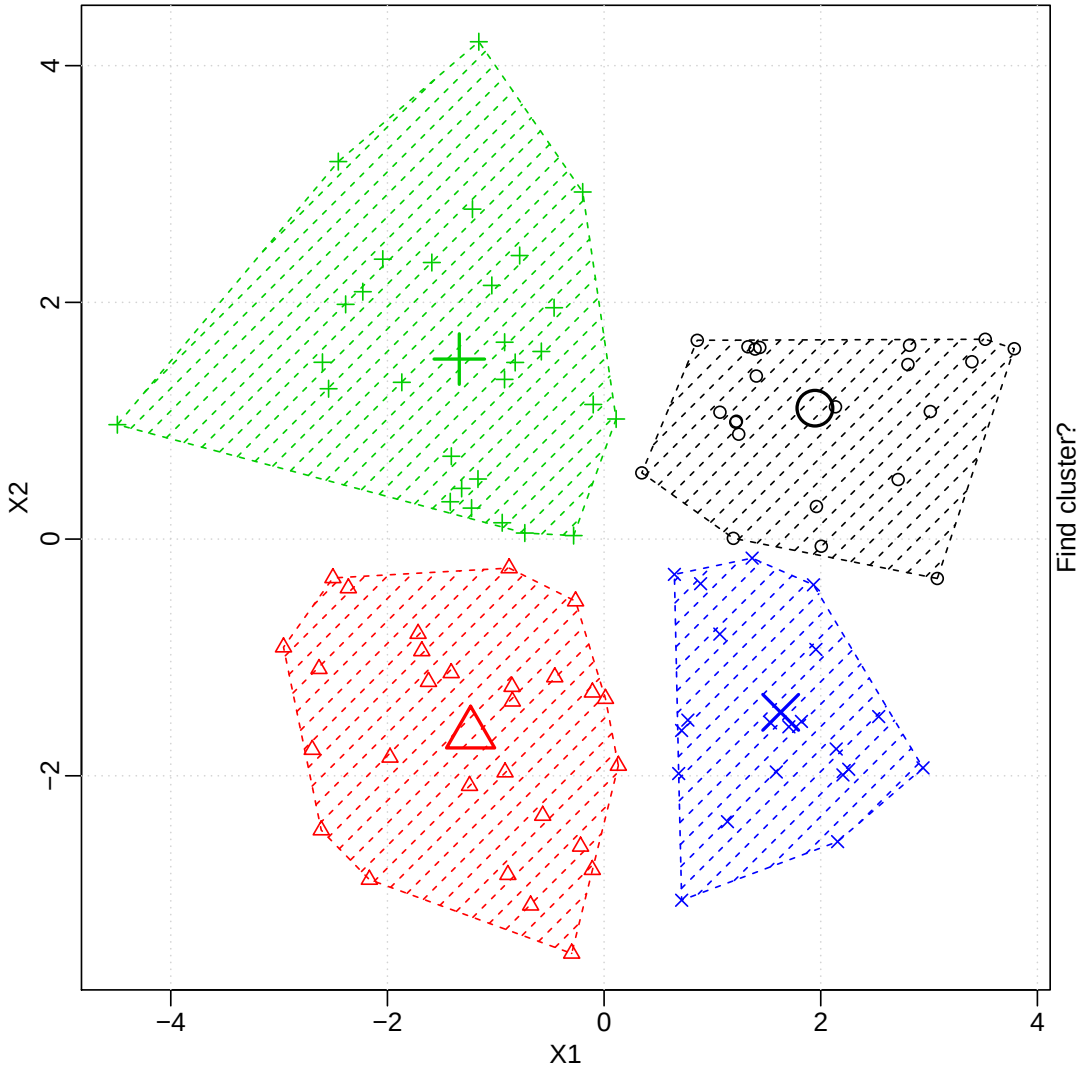

X2
X1
Move centers!
4
2
0
−2
−4
−2
0
2
4

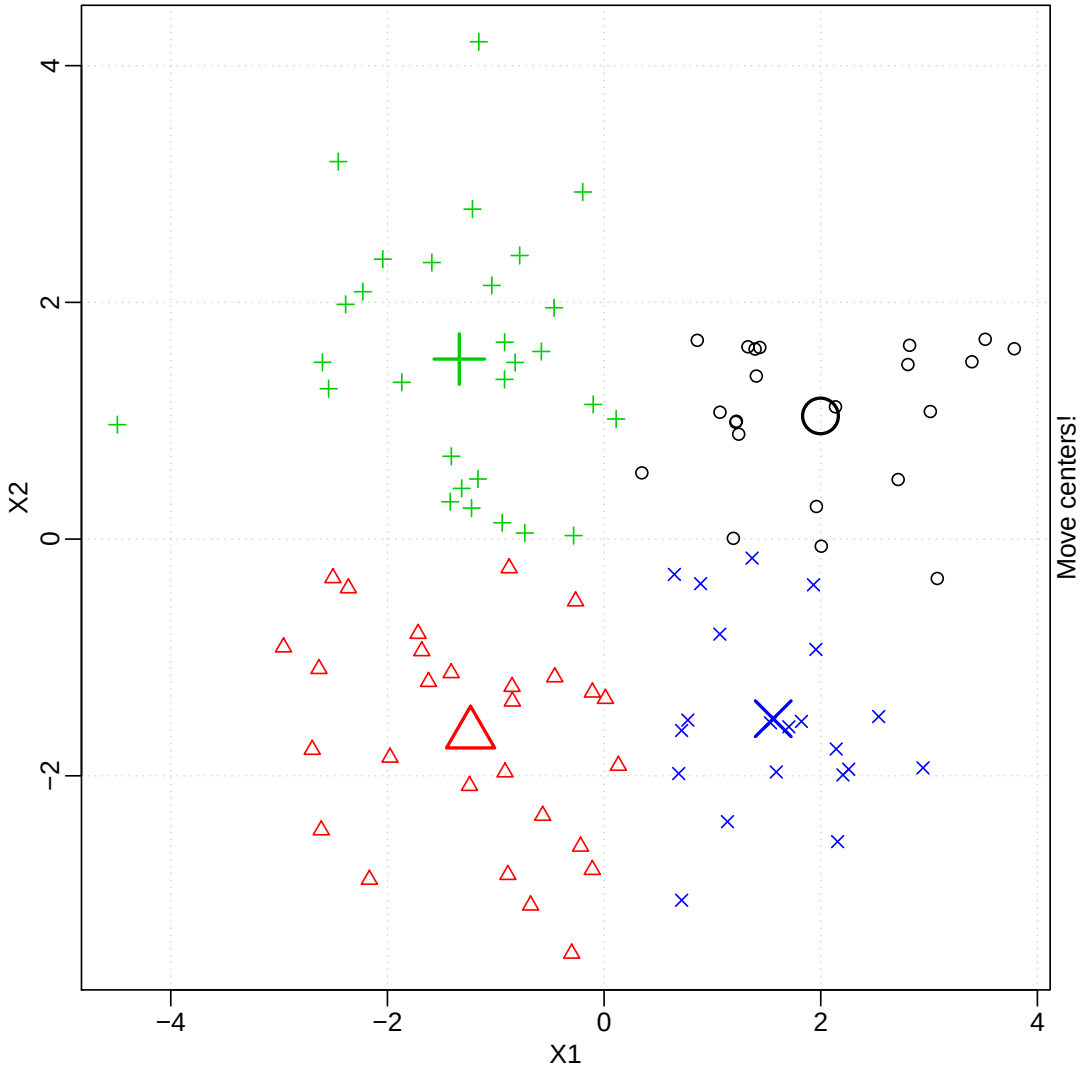

X2
4
2
0
−2
−4
−2
0
2
4
X1
Find cluster?

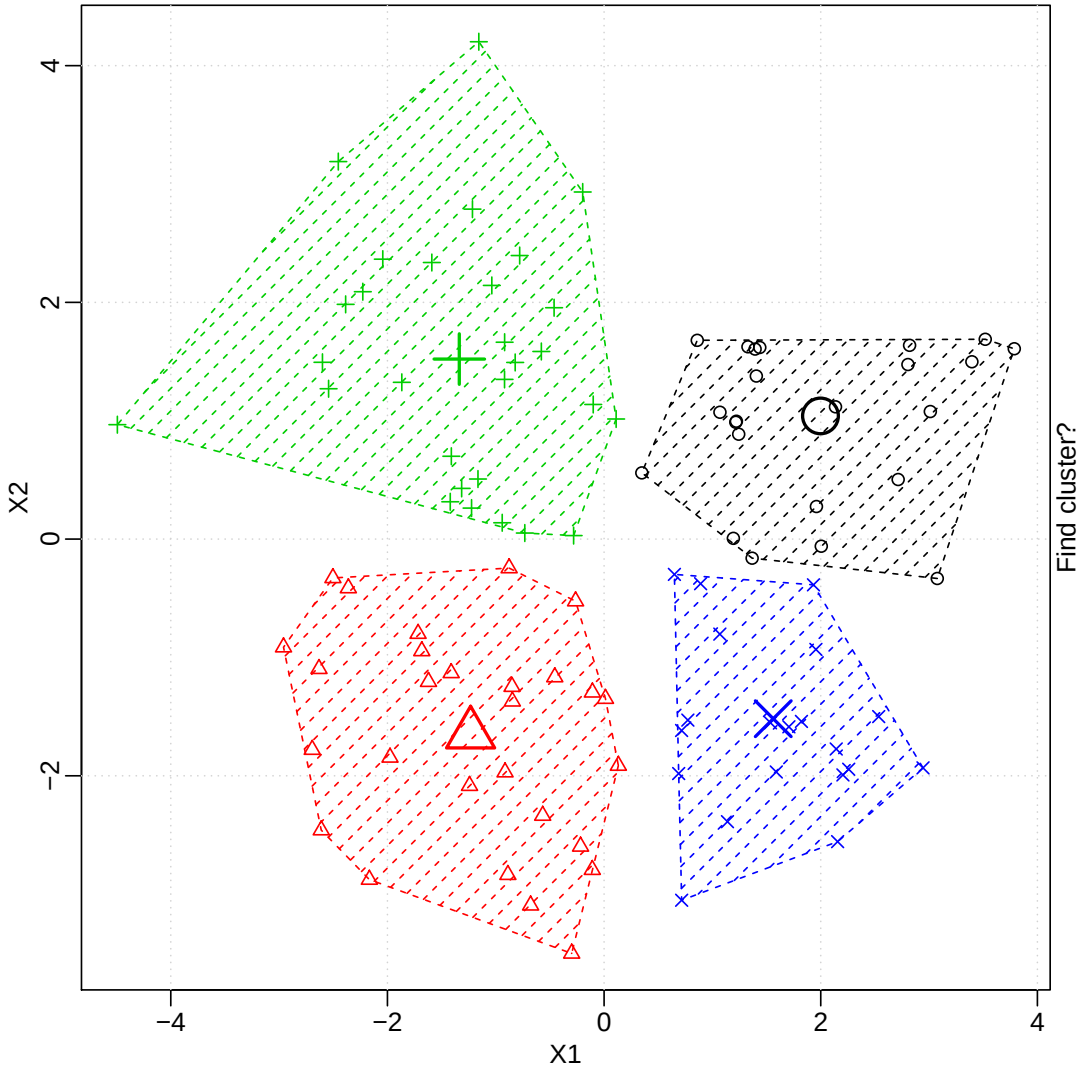

X2
X1
Move centers!
4
2
0
−2
−4
−2
0
2
4

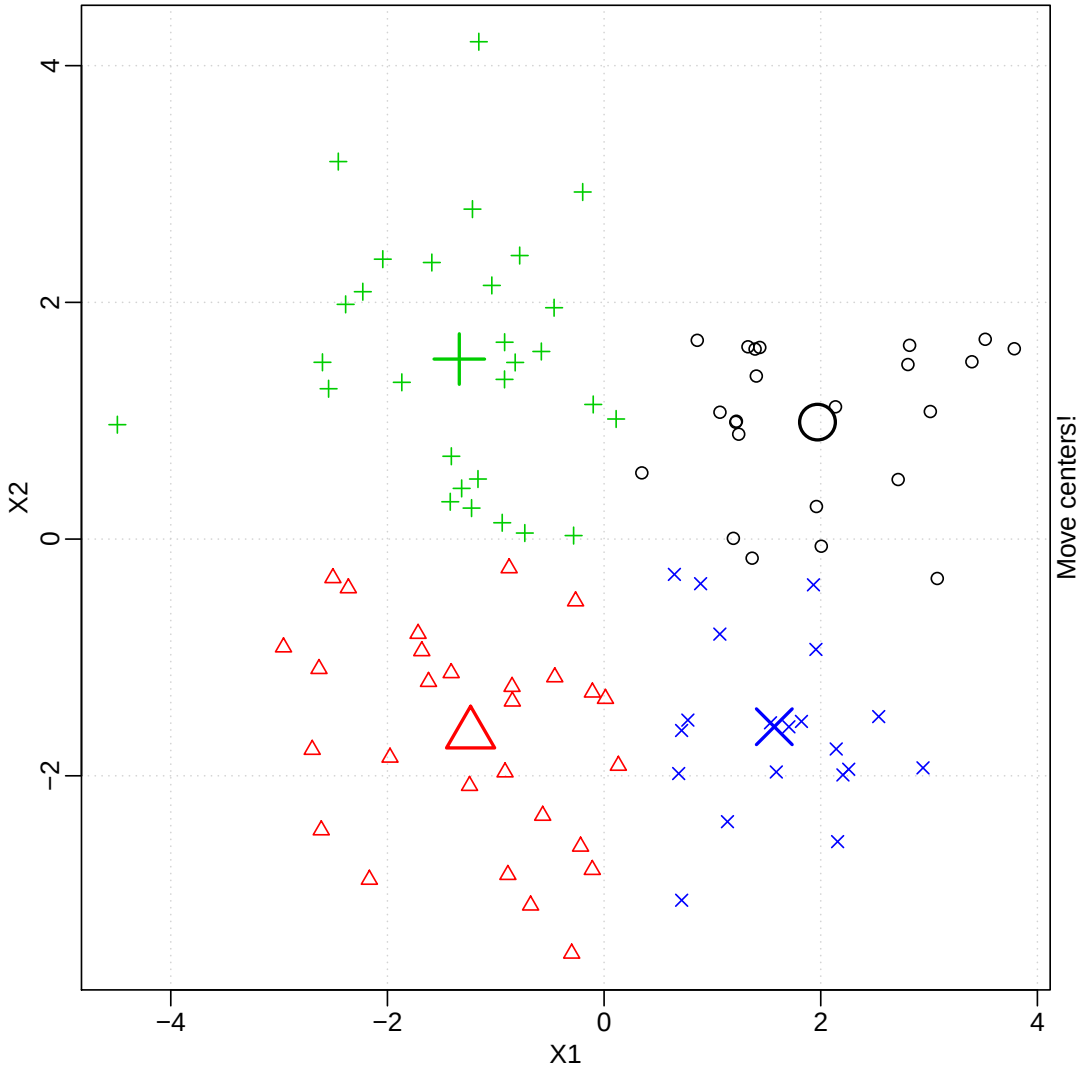

X2
X1
Find cluster?
4
2
0
−2
−4
−2
0
2
4

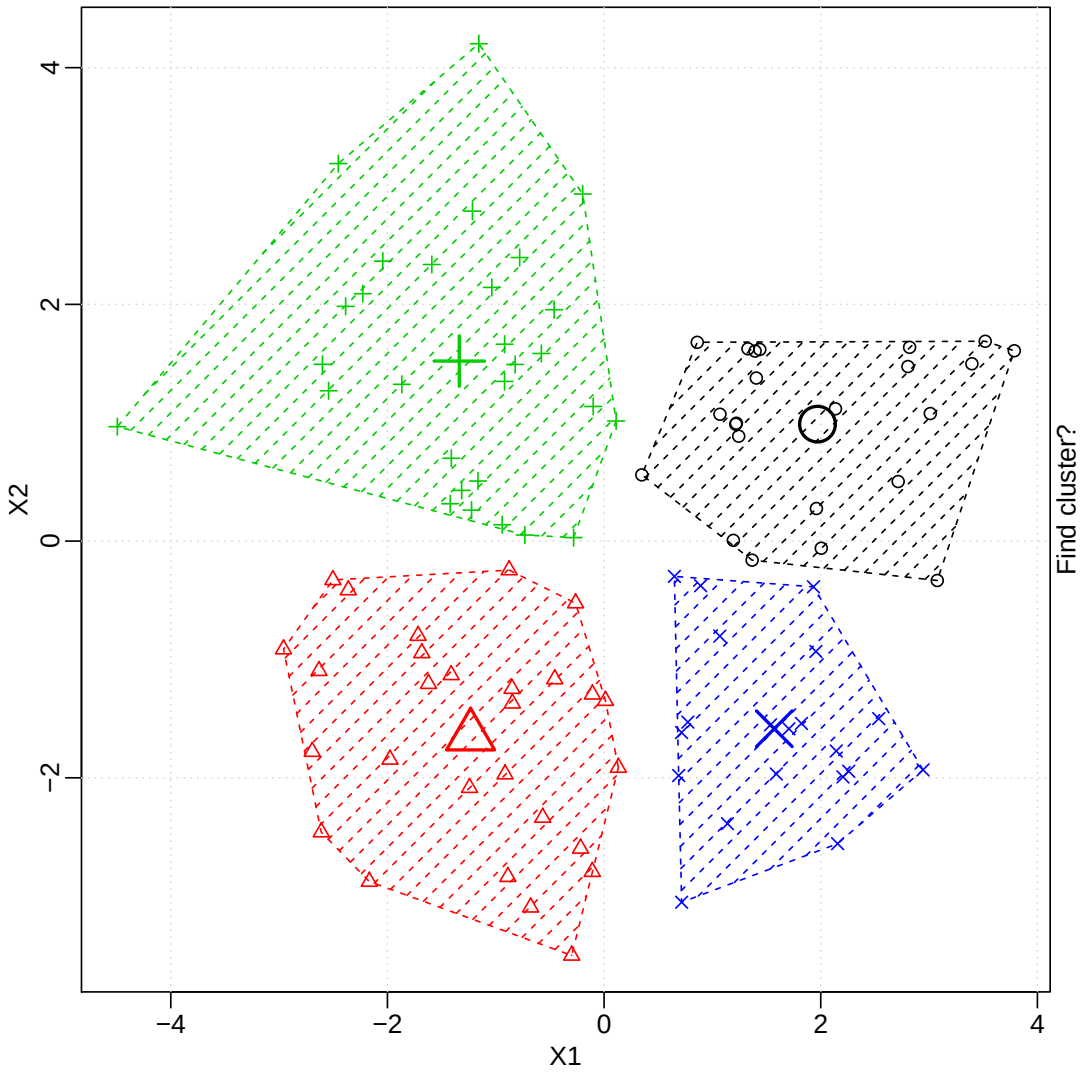

Move centers!
X1
X2
−4
−2
0
2
4
−2
0
2
4


X2
X1
Find cluster?
4
2
0
−2
−4
−2
0
2
4

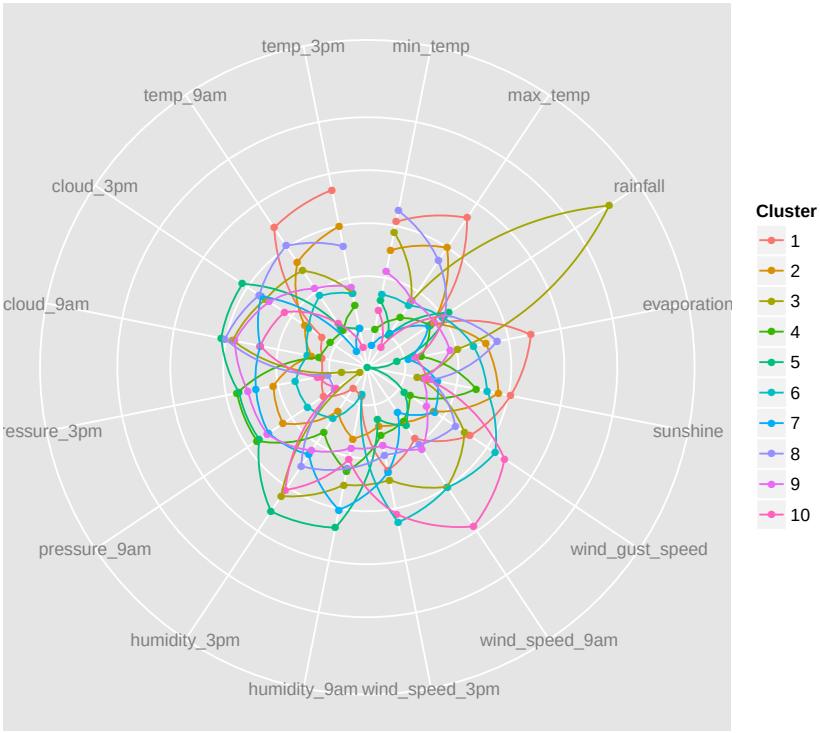

Move centers!
X1
X2
−4
−2
0
2
4
−2
0
2
4


X2
X1
Find cluster?
4
2
0
−2
−4
−2
0
2
4


X2
X1
Move centers!
4
2
0
−2
−4
−2
0
2
4

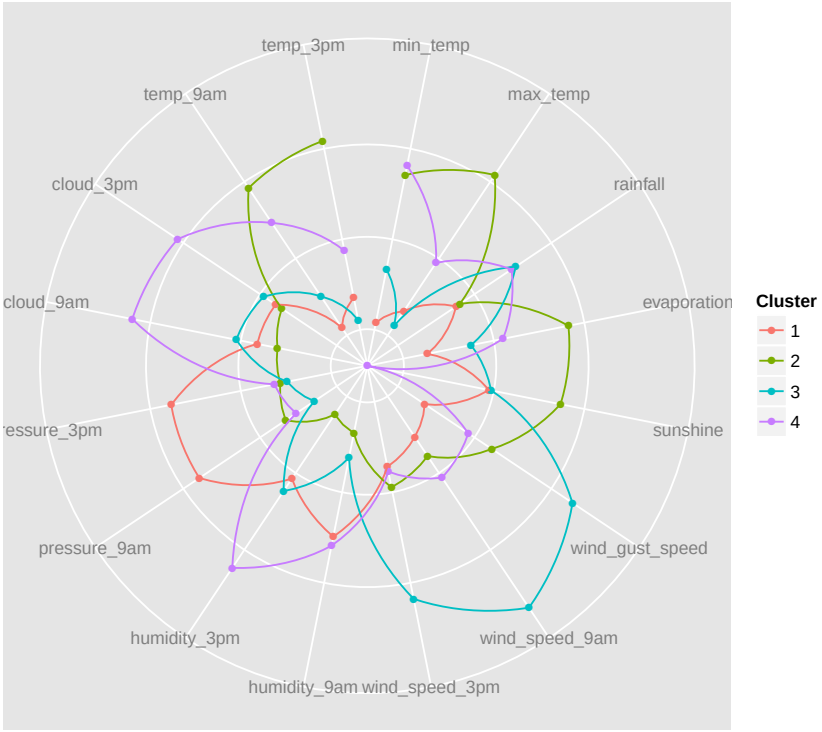

X2
4
2
0
−2
−4
−2
0
2
4
X1
Find cluster?

11 Visualise the Cluster: Radial Plot Using GGPlot2



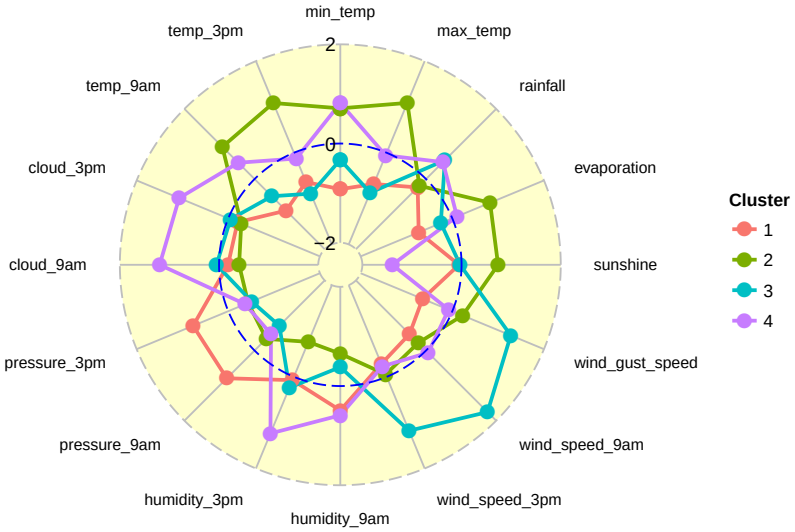

```
dscm <- melt(model$centers)
names(dscm) <- c("Cluster", "Variable", "Value")
dscm$Cluster <- factor(dscm$Cluster)
dscm$Order <- as.vector(sapply(1:length(numi), rep, 10))
p <- ggplot(subset(dscm, Cluster %in% 1:10),
            aes(x=reorder(Variable, Order),
                y=Value, group=Cluster, colour=Cluster))
p <- p + coord_polar()
p <- p + geom_point()
p <- p + geom_path()
p <- p + labs(x=NULL, y=NULL)
p <- p + theme(axis.ticks.y=element_blank(), axis.text.y = element_blank())
p
```

12 Visualize the Cluster: Radial Plot with K=4




```
nclust <- 4
model <- m.kms <- kmeans(scale(ds[numi]), nclust)
dscm <- melt(model$centers)
names(dscm) <- c("Cluster", "Variable", "Value")
dscm$Cluster <- factor(dscm$Cluster)
dscm$Order <- as.vector(sapply(1:length(numi), rep, nclust))
p <- ggplot(dscm,
            aes(x=reorder(Variable, Order),
                y=Value, group=Cluster, colour=Cluster))
p <- p + coord_polar()
p <- p + geom_point()
p <- p + geom_path()
p <- p + labs(x=NULL, y=NULL)
p <- p + theme(axis.ticks.y=element_blank(), axis.text.y = element_blank())
p
```

13 Visualise the Cluster: Cluster Profiles with Radial Plot

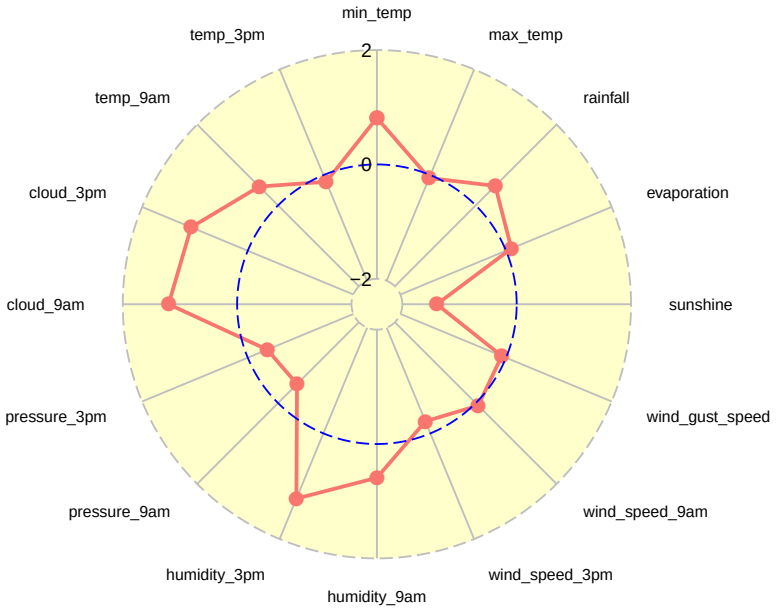


The radial plot here is carefully engineered to most effectively present the cluster profiles. The R code to generate the plot is defined as `CreateRadialPlot()` and was originally available from Paul Williamson's web site (Department of Geography, University of Liverpool):

```
source("http://onepager.togaware.com/CreateRadialPlot.R")
dsc <- data.frame(group=factor(1:4), model$centers)
CreateRadialPlot(dsc, grid.min=-2, grid.max=2, plot.extent.x=1.5)
```

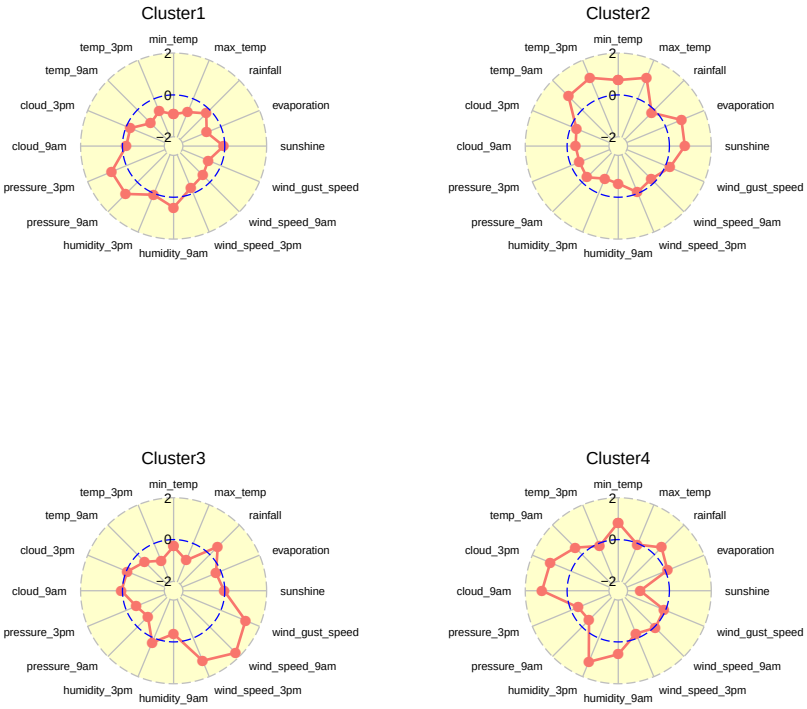
We can quickly read the profiles and gain insights into the 4 clusters. Having re-scaled all of the data we know that the "0" circle is the mean for each variable, and the range goes up to 2 standard deviations from the mean, in either direction. We observe that cluster 1 has a center with higher pressures, whilst the cluster 2 center has higher humidity and cloud cover and low sunshine, cluster 3 has high wind speeds and cluster 4 has higher temperatures, evaporation and sunshine.

14 Visualise the Cluster: Single Cluster Radial Plot




```
CreateRadialPlot(subset(dsc, group==4), grid.min=-2, grid.max=2, plot.extent.x=1.5)
```

15 Visualise the Cluster: Grid of Radial Plots




```
p1 <- CreateRadialPlot(subset(dsc, group==1),
                       grid.min=-2, grid.max=2, plot.extent.x=2)
p2 <- CreateRadialPlot(subset(dsc, group==2),
                       grid.min=-2, grid.max=2, plot.extent.x=2)
p3 <- CreateRadialPlot(subset(dsc, group==3),
                       grid.min=-2, grid.max=2, plot.extent.x=2)
p4 <- CreateRadialPlot(subset(dsc, group==4),
                       grid.min=-2, grid.max=2, plot.extent.x=2)
library(gridExtra)
grid.arrange(p1+ggtitle("Cluster1"), p2+ggtitle("Cluster2"),
             p3+ggtitle("Cluster3"), p4+ggtitle("Cluster4"))
```

16 K-Means: Base Case Cluster

```
model <- m.kms <- kmeans(scale(ds[numi]), 1)
model$size
```

```
## [1] 366
```

```
model$centers
```

```
##   min_temp  max_temp   rainfall evaporation   sunshine wind_gust_speed
## 1 9.98e-17 1.274e-16 -2.545e-16  -1.629e-16 -5.836e-16        1.99e-16
##   wind_speed_9am wind_speed_3pm humidity_9am humidity_3pm pressure_9am
## 1     -1.323e-16      -2.87e-16   -4.162e-16    -1.11e-16   -4.321e-15
....
```

```
model$totss
```

```
## [1] 5840
```

```
model$withinss
```

```
## [1] 5840
```

```
model$tot.withinss
```

```
## [1] 5840
```

```
model$betweenss
```

```
## [1] -1.819e-11
```

```
model$iter
```

```
## [1] 1
```

```
model$ifault
```

```
## NULL
```

Notice that this base case provides the centers of the original data, and the starting measure of the within sum of squares.

17 K-Means: Multiple Starts

18 K-Means: Cluster Stability

Rebuilding multiple clusterings using different random starting points will lead to different clusters being identified. We might expect that clusters that are regularly being identified with different starting points, might be more robust as actual clusters representing some cohesion among the observations belonging to that cluster.

The function `clusterboot()` from `fpc` (Hennig, 2014) provides a convenient tool to identify robust clusters.

```
library(fpc)
model <- m.kmcb <- clusterboot(scale(ds[numi]),
                               clustermethod=kmeansCBI,
                               runs=10,
                               krange=10,
                               seed=42)
```

```
## boot 1
## boot 2
## boot 3
## boot 4
....
```

```
model
```

```
## * Cluster stability assessment *
## Cluster method:  kmeans
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
....
```

```
str(model)
```

```
## List of 31
##  $ result       :List of 6
##   ..$ result       :List of 11
##   .. ..$ cluster     : int [1:366] 1 10 5 7 3 1 1 1 1 1 ...
....
```

19 Evaluation of Clustering Quality

Numerous measures are available for evaluating a clustering. Many are stored within the data structure returned by `kmeans()`

The total sum of squares.

```
model <- kmeans(scale(ds[numi]), 10)
model$totss
```

```
## [1] 5840
```

```
model$withinss
```

```
##  [1] 172.1 219.2 237.6 217.2 336.6 228.4 254.2 291.9 310.5 126.0
```

```
model$tot.withinss
```

```
## [1] 2394
```

```
model$betweenss
```

```
## [1] 3446
```

The basic concept is the sum of squares. This is typically a sum of the square of the distances between observations.

20 Evaluation: Within Sum of Squares

The within sum of squares is a measure of how close the observations are within the clusters. For a single cluster this is calculated as the average squared distance of each observation within the cluster from the cluster mean. Then the total within sum of squares is the sum of the within sum of squares over all clusters.

The total within sum of squares generally decreases as the number of clusters increases. As we increase the number of clusters they individually tend to become smaller and the observations closer together within the clusters. As k increases, the changes in the total within sum of squares would be expected to reduce, and so it flattens out. A good value of k might be where the reduction in the total weighted sum of squares begins to flatten.

```
model$withinss
```

```
##  [1] 172.1 219.2 237.6 217.2 336.6 228.4 254.2 291.9 310.5 126.0
```

```
model$tot.withinss
```

```
## [1] 2394
```

The total within sum of squares is a common measure that we aim to minimise in building a clustering.

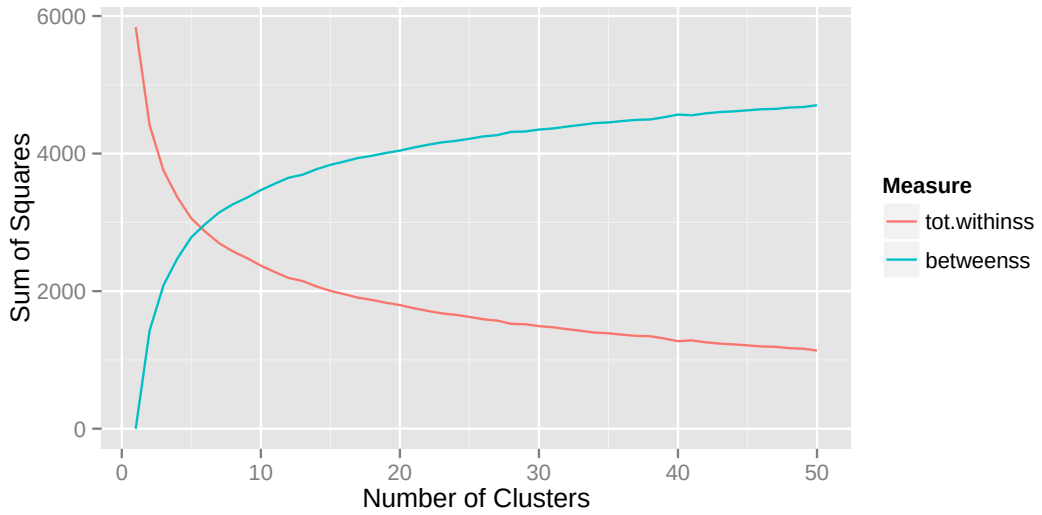
21 Evaluation: Between Sum of Squares

The between sum or squares is a measure of how far the clusters are from each other.

```
model$betweenss
```

```
## [1] 3446
```

A good clustering will have a small within sum of squares and a large between sum of squares. Here we see the relationship between these two measures:

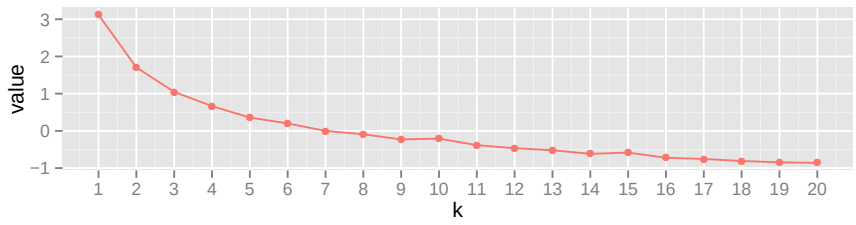


22 K-Means: Selecting k Using Scree Plot

```
crit  <- vector()
nk    <- 1:20
for (k in nk)
{
  m <- kmeans(scale(ds[numi]), k)
  crit <- c(crit, sum(m$withinss))
}
crit
```

```
##  [1] 5840 4414 3753 3368 3057 2900 2697 2606 2465 2487 2310 2228 2173 2075
## [15] 2108 1970 1937 1882 1846 1830
```

```
dsc  <- data.frame(k=nk, crit=scale(crit))
dscm <- melt(dsc, id.vars="k", variable.name="Measure")

p <- ggplot(dscm, aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure))
p <- p + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p <- p + theme(legend.position="none")
p
```



23 K-Means: Selecting k Using Calinski-Harabasz

The Calinski-Harabasz criteria, also known as the variance ratio criteria, is the ratio of the *between sum of squares* (divided by $k-1$) to the *within sum of squares* (divided by $n-k$)—the sum of squares is a measure of the variance. The relative values can be used to compare clusterings of a single dataset, with higher values being better clusterings. The criteria is said to work best for spherical clusters with compact centres (as with normally distributed data) using k-means with Euclidean distance.

```
library(fpc)
nk <- 1:20
model <- km.c <- kmeansruns(scale(ds[numi]), krange=nk, criterion="ch")
class(model)
```

```
## [1] "kmeans"
```

```
model
```

```
## K-means clustering with 2 clusters of sizes 192, 174
##
## Cluster means:
##   min_temp max_temp rainfall evaporation sunshine wind_gust_speed
....
```

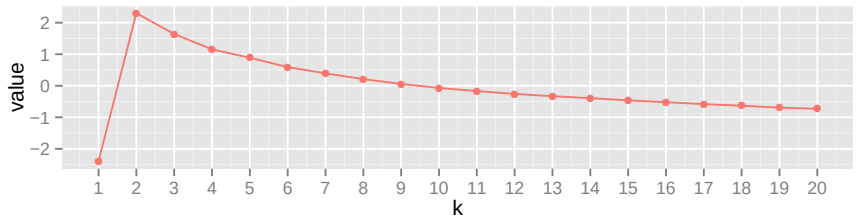
```
model$crit
```

```
##  [1]   0.00 117.55 100.97  88.81  82.16  74.75  69.75  65.18  61.38  58.18
## [11]  55.71  53.44  51.63  50.07  48.34  46.90  45.32  44.07  42.57  41.65
```

```
model$bestk
```

```
## [1] 2
```

```
dsc  <- data.frame(k=nk, crit=scale(km.c$crit))
dscm <- melt(dsc, id.vars="k", variable.name="Measure")

p <- ggplot(dscm, aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure))
p <- p + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p <- p + theme(legend.position="none")
p
```



24 K-Means: Selecting k Using Average Silhouette Width

The average silhouette width criteria is more computationally expensive than the Calinski-Harabasz criteria which is an issue for larger datasets. A dataset of 50,000 observations and 15 scaled variables, testing from 10 to 40 clusters, 10 runs, took 30 minutes for the Calinski-Harabasz criteria compared to minutes using the average silhouette width criteria.

check timing

```
library(fpc)
nk <- 1:20
model <- km.a <- kmeansruns(scale(ds[numi]), krange=nk, criterion="asw")
class(model)
```

```
## [1] "kmeans"
```

```
model
```

```
## K-means clustering with 2 clusters of sizes 174, 192
##
## Cluster means:
##   min_temp max_temp rainfall evaporation sunshine wind_gust_speed
....
```

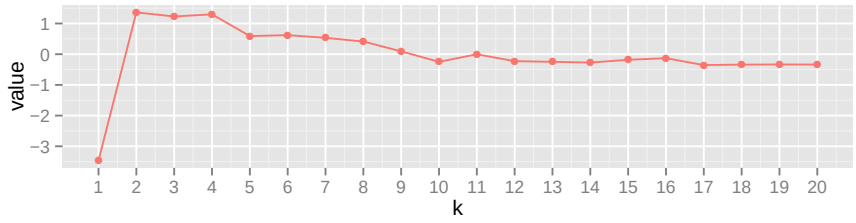
```
model$crit
```

```
##  [1] 0.0000 0.2255 0.2192 0.2225 0.1894 0.1908 0.1867 0.1811 0.1662 0.1502
## [11] 0.1617 0.1512 0.1502 0.1490 0.1533 0.1557 0.1453 0.1459 0.1462 0.1460
```

```
model$bestk
```

```
## [1] 2
```

```
dsc  <- data.frame(k=nk, crit=scale(km.a$crit))
dscm <- melt(dsc, id.vars="k", variable.name="Measure")

p <- ggplot(dscm, aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure))
p <- p + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p <- p + theme(legend.position="none")
p
```



25 K-Means: Using clusterCrit Calinski_Harabasz

The clusterCrit (Desgraupes, 2013) package provides a comprehensive collection of clustering criteria. Here we illustrate its usage with the Calinski_Harabasz criteria. Do note that we obtain a different model here to that above, hence different calculations of the criteria.

```
library(clusterCrit)
crit <- vector()
for (k in 1:20)
{
  m <- kmeans(scale(ds[numi]), k)
  crit <- c(crit, as.numeric(intCriteria(as.matrix(ds[numi]), m$cluster,
                                         "Calinski_Harabasz")))
}
crit[is.nan(crit)] <- 0 #
crit
```

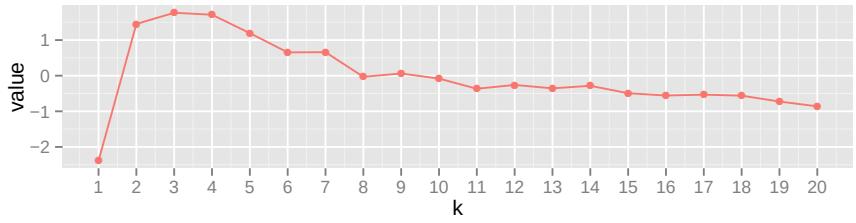
```
##  [1]  0.00 81.20 87.88 86.71 75.78 64.34 64.48 49.87 51.83 48.81 42.78
## [12] 45.04 43.03 44.53 40.12 38.76 39.26 38.67 35.19 32.33
```

```
bestCriterion(crit, "Calinski_Harabasz")
```

```
## [1] 3
```

In this case $k = 3$ is the optimum choice.

```
dsc  <- data.frame(k=nk, crit=scale(crit))
dscm <- melt(dsc, id.vars="k", variable.name="Measure")

p <- ggplot(dscm, aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure))
p <- p + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p <- p + theme(legend.position="none")
p
```



26 K-Means: Compare All Criteria

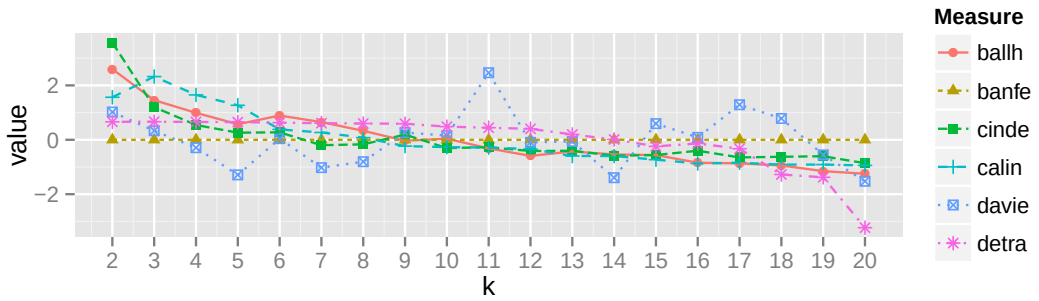
We can generate all criteria and then plot them. There are over 40 criteria, and the are noted on the help page for `intCriteria()`. We generate all the criteria here and then plot the first 6 below, with the remainder in the following section.

```
m <- kmeans(scale(ds[numi]), 5)
ic <- intCriteria(as.matrix(ds[numi]), m$cluster, "all")
names(ic)
```

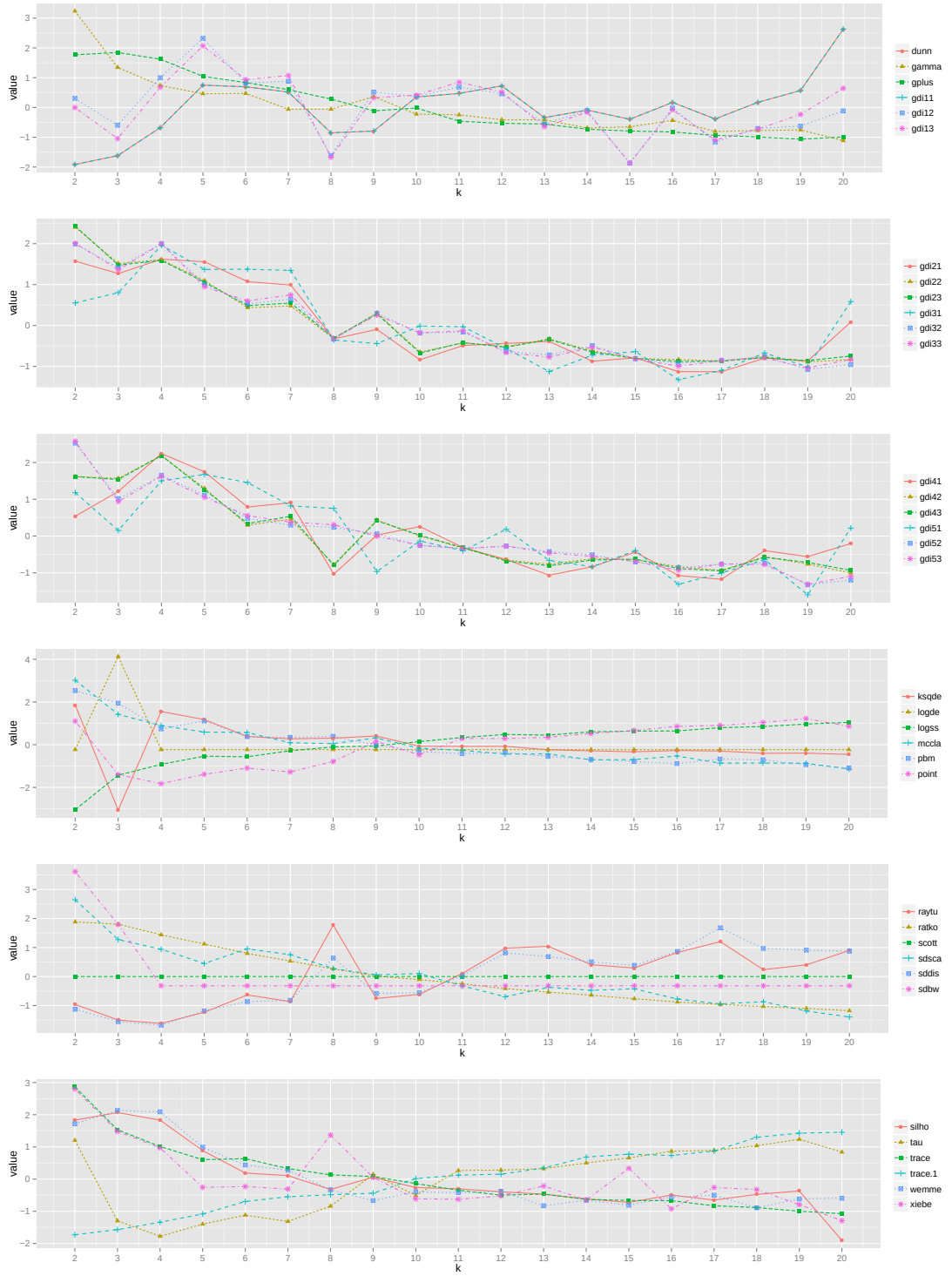
```
##  [1] "ball_hall"         "banfeld_raftery"   "c_index"
##  [4] "calinski_harabasz" "davies_bouldin"    "det_ratio"
##  [7] "dunn"              "gamma"             "g_plus"
## [10] "gdi11"             "gdi12"             "gdi13"
....
```

```
crit <- data.frame()
for (k in 2:20)
{
  m <- kmeans(scale(ds[numi]), k)
  crit <- rbind(crit, as.numeric(intCriteria(as.matrix(ds[numi]), m$cluster,
                                             "all")))
}
names(crit) <- substr(sub("_", "", names(ic)), 1, 5) # Shorten for plots.
crit <- data.frame(sapply(crit, function(x) {x[is.nan(x)] <- 0; x}))
```

```
dsc  <- cbind(k=2:20, data.frame(sapply(crit, scale)))

dscm <- melt(dsc, id.vars="k", variable.name="Measure")
dscm$value[is.nan(dscm$value)] <- 0

ms <- as.character(unique(dscm$Measure))

p <- ggplot(subset(dscm, Measure %in% ms[1:6]), aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure)) + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p
```



27 K-Means: Plot All Criteria



28 K-Means: predict()

rattle (Williams, 2014) provides a `predict.kmeans()` to assign new observations to their nearest means.

```
set.seed(42)
train <- sample(nobs, 0.7*nobs)
test  <- setdiff(seq_len(nobs), train)

model <- kmeans(ds[train, numi], 2)
predict(model, ds[test, numi])
```

```
##   4   5   6   8  11  14  15  16  17  21  28  30  32  36  44  47  50  55
## "2" "2" "2" "2" "1" "1" "1" "1" "1" "1" "1" "2" "1" "2" "1" "1" "2" "2"
##  57  61  62  63  67  74  75  76  77  80  84  92  94  95  97  98  99 100
## "2" "1" "1" "1" "1" "1" "2" "1" "1" "2" "1" "1" "1" "1" "2" "1" "2" "2"
....
```

29 Entropy Weighted K-Means

Sometimes it is better to build clusters based on subsets of variables, particularly if there are many variables. Subspace clustering and bicluster analysis are approaches to doing this.

We illustrate the concept here using wskm (Williams *et al.*, 2012), for weighted subspace k-means. We use the `ewkm()` (entropy weighted k-means).

```
set.seed(42)
library(wskm)
m.ewkm <- ewkm(ds, 10)
```

```
## Warning:  NAs introduced by coercion
```

```
## Error:  NA/NaN/Inf in foreign function call (arg 1)
```

The error is expected and once again only numeric variables can be clustered.

```
m.ewkm <- ewkm(ds[numi], 10)
```

```
## **********Clustering converged. Terminate!
```

```
round(100*m.ewkm$weights)
```

```
##    min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## 1         0        0      100           0        0               0
## 2         0        0        0         100        0               0
## 3         0        0      100           0        0               0
## 4         0        0        0           0        0               0
## 5         6        6        6           6        6               6
## 6         0        0        0         100        0               0
## 7         0        0        0         100        0               0
## 8         0        0        0           0        0               0
## 9         6        6        6           6        6               6
## 10        0        0      100           0        0               0
....
```

Exercise: Plot the clusters.

Exercise: Rescale the data so all variables have the same range and then rebuild the cluster, and comment on the differences.

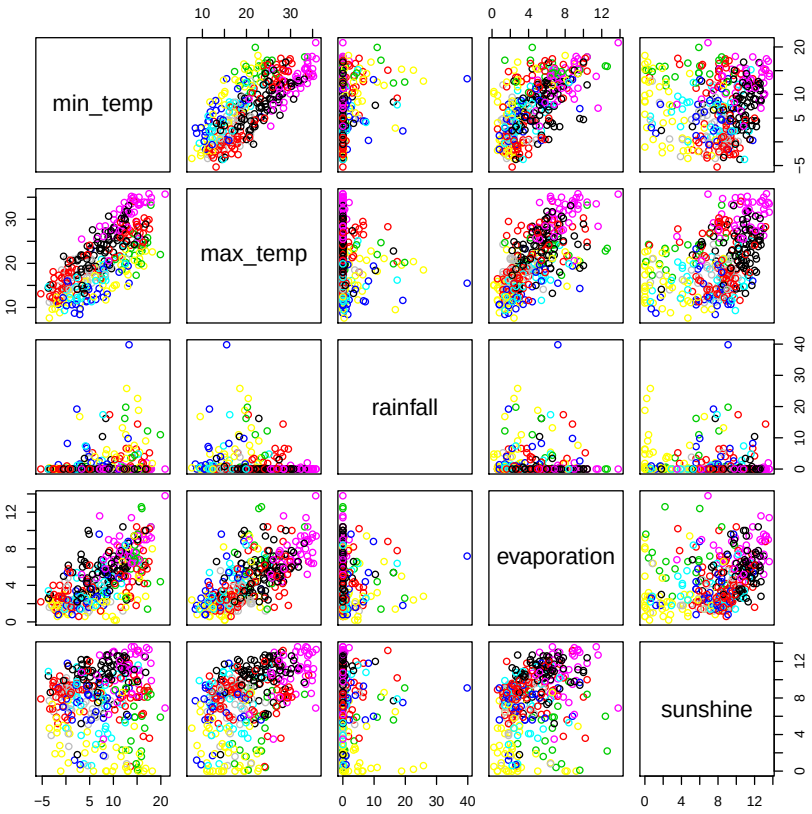
Exercise: Discuss why ewkm might be better than k-means. Consider the number of variables as an advantage, particularly in the context of the curse of dimensionality.

30 Partitioning Around Medoids: PAM

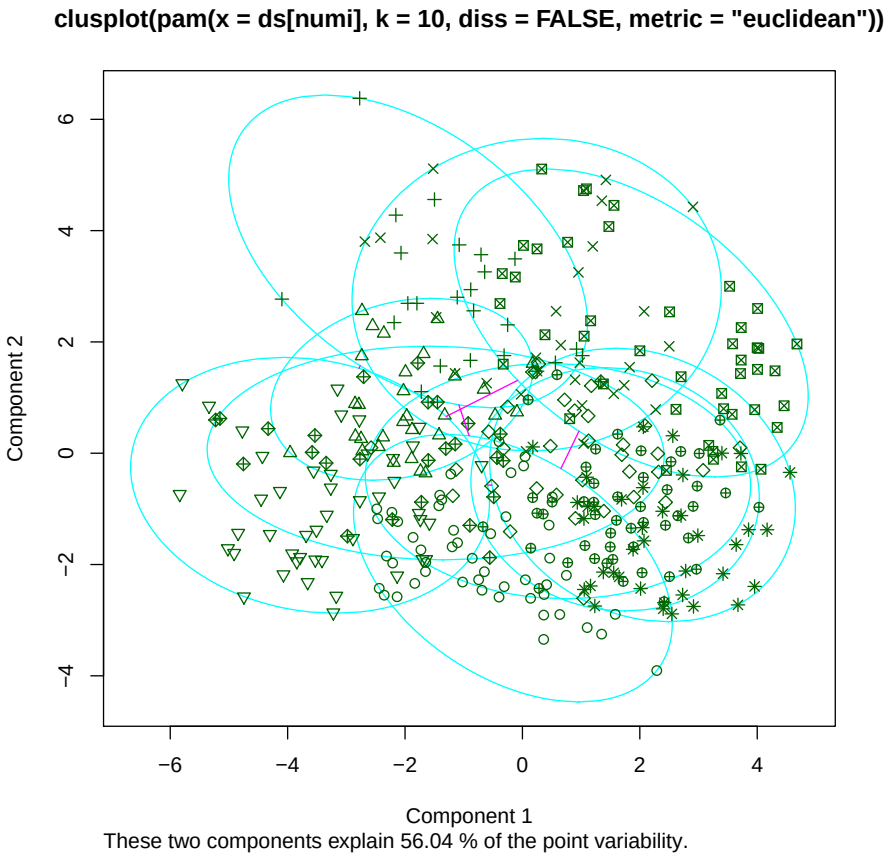
```
model <- pam(ds[numi], 10, FALSE, "euclidean")

summary(model)
```

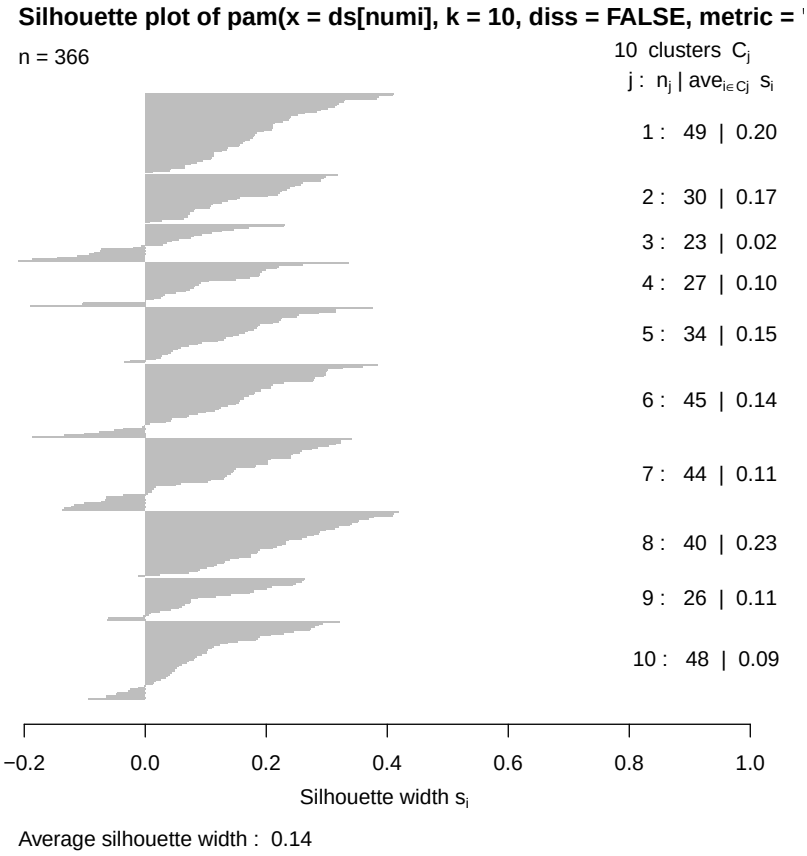
```
## Medoids:
##        ID min_temp max_temp rainfall evaporation sunshine wind_gust_speed
##  [1,]  11      9.1     25.2      0.0         4.2     11.9              30
##  [2,]  38     16.5     28.2      4.0         4.2      8.8              39
....
```

```
plot(ds[numi[1:5]], col=model$clustering)
points(model$medoids, col=1:10, pch=4)
```




```
plot(model)
```


clusplot(pam(x = ds[numi], k = 10, diss = FALSE, metric = "euclidean"))
Component 2
Component 1
These two components explain 56.04 % of the point variability.


Silhouette plot of pam(x = ds[numi], k = 10, diss = FALSE, metric =
n = 366
10 clusters C_j
j : n_j | $ave_{i \in Cj}$ s_i
1 : 49 | 0.20
2 : 30 | 0.17
3 : 23 | 0.02
4 : 27 | 0.10
5 : 34 | 0.15
6 : 45 | 0.14
7 : 44 | 0.11
8 : 40 | 0.23
9 : 26 | 0.11
10 : 48 | 0.09
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Silhouette width s_i
Average silhouette width : 0.14

31 Clara

32 Hierarchical Cluster in Parallel

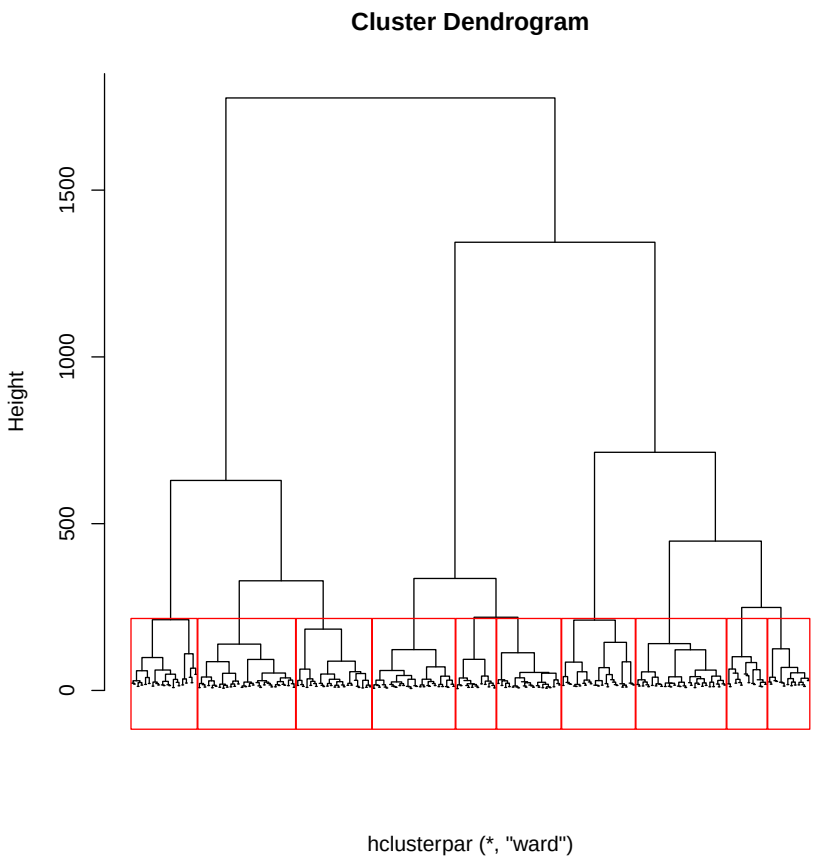
Use `hclusterpar()` from `amap` (Lucas, 2011).

```
library(amap)
model <- hclusterpar(na.omit(ds[numi]),
                     method="euclidean",
                     link="ward",
                     nbproc=1)
```

33 Plotting Hierarchical Cluster

Plot from cba (Buchta and Hahsler, 2014).

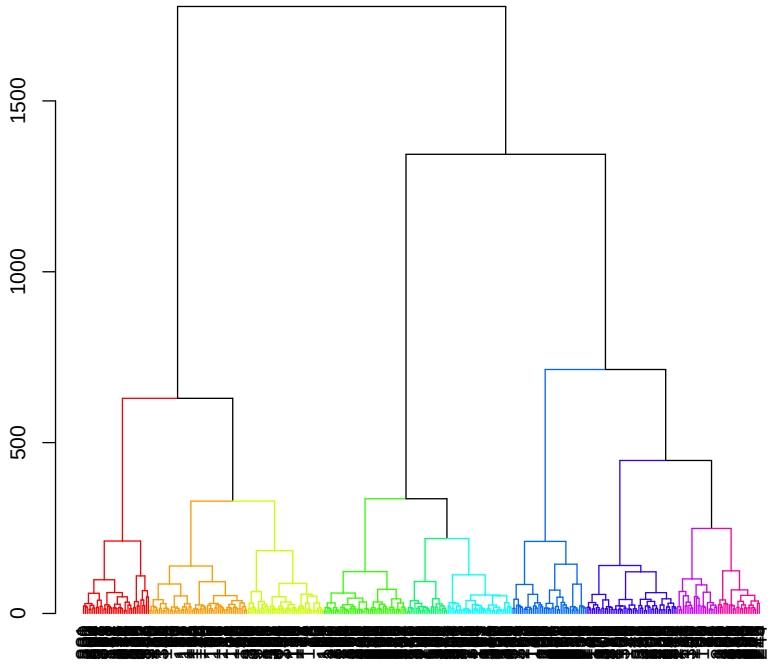
```
plot(model, main="Cluster Dendrogram", xlab="", labels=FALSE, hang=0)

#Add in rectangles to show the clusters.

rect.hclust(model, k=10)
```



34 Add Colour to the Hierarchical Cluster

Using the dendroextras (Jefferis, 2014) package to add colour to the dendrogram:

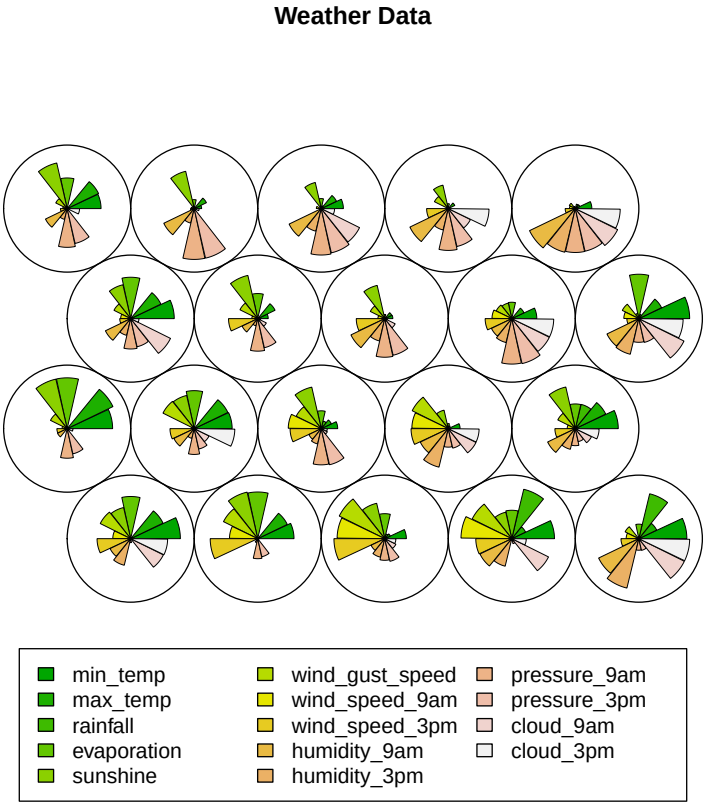
```
library(dendroextras)
plot(colour_clusters(model, k=10), xlab="")
```



35 Hierarchical Cluster Binary Variables

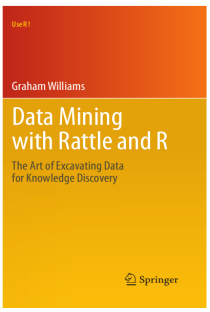
Exercise: Clustering a large population based on the patterns of missing data within the population is a technique for grouping observations exhibiintg similar patterns of behaviouour assuming missing by pattern.... We can convert each variable to a binary 1/0 indicating present/missing and then use mona() for a hiearchical clustering. Demonstrate this. Include a levelplot.

36 Self Organising Maps: SOM




```
library("kohonen")
set.seed(42)
model <- som(scale(ds[numi[1:14]]), grid = somgrid(5, 4, "hexagonal"))
plot(model, main="Weather Data")
```

37 Further Reading and Acknowledgements



The Rattle Book, published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from Amazon. Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from `http://datamining.togaware.com`, including the Datamining Desktop Survival Guide.

This module is one of many OnePageR modules available from `http://onepager.togaware.com`. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.

Other resources include:

- Practical Data Science with R by Nina Zumel and John Mount, March 2014, has a good chapter on Cluster Analysis with some depth of explanation of the sum of squares measures, and good examples of R code for performing cluster analysis. It also covers `clusterboot()` and `kmeansruns()` from fpc.
- The radar or radial plot code originated from an RStudio Blog Posting.
- The definition of all criteria used to measure the goodness of a clustering can be found in a Vingette of the clusterCrit (Desgraupes, 2013) package. Also available on CRAN.

38 References

Breiman L, Cutler A, Liaw A, Wiener M (2012). *randomForest: Breiman and Cutler's random forests for classification and regression*. R package version 4.6-7, URL http://CRAN.R-project.org/package=randomForest.

Buchta C, Hahsler M (2014). *cba: Clustering for Business Analytics*. R package version 0.2-14, URL http://CRAN.R-project.org/package=cba.

Desgraupes B (2013). *clusterCrit: Clustering Indices*. R package version 1.2.3, URL http://CRAN.R-project.org/package=clusterCrit.

Hennig C (2014). *fpc: Flexible procedures for clustering*. R package version 2.1-7, URL http://CRAN.R-project.org/package=fpc.

Jefferis G (2014). *dendroextras: Extra functions to cut, label and colour dendrogram clusters*. R package version 0.1-4, URL http://CRAN.R-project.org/package=dendroextras.

Lucas A (2011). *amap: Another Multidimensional Analysis Package*. R package version 0.8-7, URL http://CRAN.R-project.org/package=amap.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Williams GJ (2009). "Rattle: A Data Mining GUI for R." *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.

Williams GJ (2014). *rattle: Graphical user interface for data mining in R*. R package version 3.0.4, URL http://rattle.togaware.com/.

Williams GJ, Huang JZ, Chen X, Wang Q, Xiao L (2012). *wskm: Weighted k-means Clustering*. R package version 1.4.0, URL http://CRAN.R-project.org/package=wskm.

Xie Y (2013). *animation: A gallery of animations in statistics and utilities to create animations*. R package version 2.2, URL http://CRAN.R-project.org/package=animation.

This document, sourced from ClustersO.Rnw revision 440, was processed by KnitR version 1.6 of 2014-05-24 and took 30.6 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-06-22 12:38:37.