Clustering

Rob McCulloch

1. Undirected Learning/Data Mining
2. The Cereal Data
3. Distance
4. Hierarchical Clustering
5. K-means Clustering

1. Undirected Learning/Data Mining

Up until now we have emphasized directed data mining.

This is the x and y game.

Given x, what's y ?????

Now we will look at tools in undirected data mining.

We will start with the most fundamental one, clustering.

What is undirected data mining ?

We still have observations on several variables,
and we are still looking for some kind of pattern.

But now, there is no “y”.

We just have “x” and want to see if there is structure.

3

For example, suppose we have two numeric variables x_1 and x_2.

If I compute the correlation between x_1 and x_2 I could say I am doing "undirected data mining/learning".

If I regress, x_2 on x_1, with the goal of predicting x_2 given future x_1 values, I am doing directed data/learning mining.

The basic technique in undirected DM is clustering.

We take our observations and try to divide them into groups of customers or brands or whatever.

It can be simpler to understand a large set (eg of customers) by saying we have a group like that, and a group like that, and so on.

Maybe our models will work better if you do one group at a time.

5

"... most of the data mining projects going on in the real world are directed".

Berry and Linoff

Directed data mining is also called *supervised data mining* or *supervised learning.*

And undirected data mining is also called, *unsupervised learing.*

6

"With supervised learning there is a clear measure of success.... This can be estimated in a variety of ways including cross-validation. In the context of unsupervised learning, there is no such direct measure of success. It is difficult to ascertain the validity of inferences drawn from the output of most unsupervised learning algorithms. One must resort to heuristic arguments not only for motivating the algorithms, as is often the case in supervised learning as well, but also for judgments as to the quality of the results. This uncomfortable situation has led to heavy proliferation of proposed methods, since effectiveness is a matter of opinion and cannot be verified directly".

Hastie, Tibshirani, and Friedman, page 439

2. The Cereal Data

We have data on different brands of cereal.

For each brand we have different measures of product characteristics.

```
> dim(cereal)
[1] 43 8
```

We have 43 brands.
For each brand we have measurements
on 8 characteristics.

```
> row.names(cereal)
 [1] "ACCheerios"             "Cheerios"
 [3] "CocoaPuffs"             "CountChocula"
 [5] "GoldenGrahams"          "HoneyNutCheerios"
 [7] "Kix"                    "LuckyCharms"
 [9] "MultiGrainCheerios"     "OatmealRaisinCrisp"
[11] "RaisinNutBran"          "TotalCornFlakes"
[13] "TotalRaisinBran"        "TotalWholeGrain"
[15] "Trix"                   "Cheaties"
[17] "WheatiesHoneyGold"      "AllBran"
[19] "AppleJacks"             "CornFlakes"
[21] "CornPops"               "CracklinOatBran"
[23] "Crispix"                "FrootLoops"
[25] "FrostedFlakes"          "FrostedMiniWheats"
[27] "FruitfulBran"           "JustRightCrunchyNuggets"
[29] "MueslixCrispyBlend"     "NutNHoneyCrunch"
[31] "NutriGrainAlmondRaisin" "NutriGrainWheat"
[33] "Product19"              "RaisinBran"
[35] "RiceKrispies"           "Smacks"
[37] "SpecialK"               "CapNCrunch"
[39] "HoneyGrahamOhs"         "Life"
[41] "PuffedRice"             "PuffedWheat"
[43] "QuakerOatmeal"
```

The Brands.

The variables:

```
> names(cereal)
[1] "calories" "protein"  "fat"      "sodium"   "fiber"
"carbo"    "sugar"
[8] "potass"
```

Thus, each variable is a characteristic of the brand.

This is a common type of application.

Another common kind of data has consumer responses to various questions about the brands as the variables.

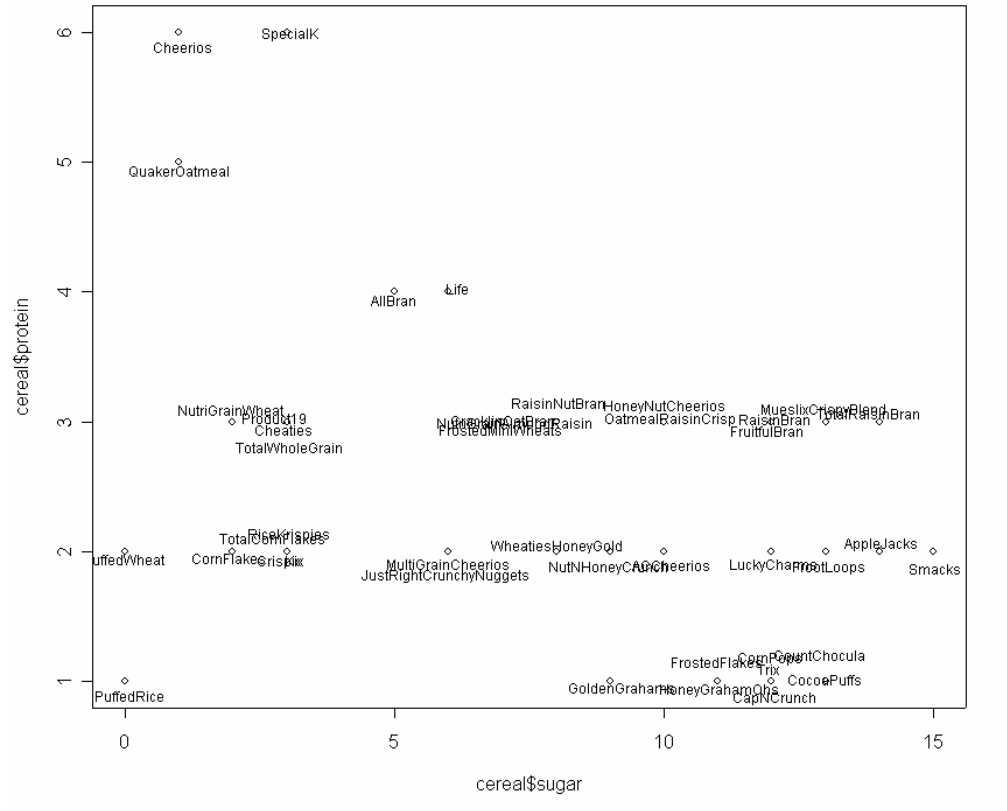
Other than calories, everything is in grams or milligrams.

```
> summary(cereal)
    calories        protein           fat              sodium      
 Min.   : 50.0   Min.   :1.000   Min.   :0.0000   Min.   :  0.0  
 1st Qu.:100.0   1st Qu.:2.000   1st Qu.:0.0000   1st Qu.:145.0  
 Median :110.0   Median :2.000   Median :1.0000   Median :190.0  
 Mean   :107.9   Mean   :2.465   Mean   :0.9767   Mean   :180.5  
 3rd Qu.:110.0   3rd Qu.:3.000   3rd Qu.:1.5000   3rd Qu.:220.0  
 Max.   :160.0   Max.   :6.000   Max.   :3.0000   Max.   :320.0  
     fiber            carbo           sugar            potass      
 Min.   :0.000   Min.   : 0.00   Min.   : 0.000   Min.   : 15.00  
 1st Qu.:0.500   1st Qu.:12.00   1st Qu.: 3.000   1st Qu.: 37.50  
 Median :1.000   Median :14.00   Median : 8.000   Median : 60.00  
 Mean   :1.714   Mean   :14.01   Mean   : 7.605   Mean   : 84.42  
 3rd Qu.:2.850   3rd Qu.:17.00   3rd Qu.:12.000   3rd Qu.:110.00  
 Max.   :9.000   Max.   :22.00   Max.   :15.000   Max.   :320.00  
```

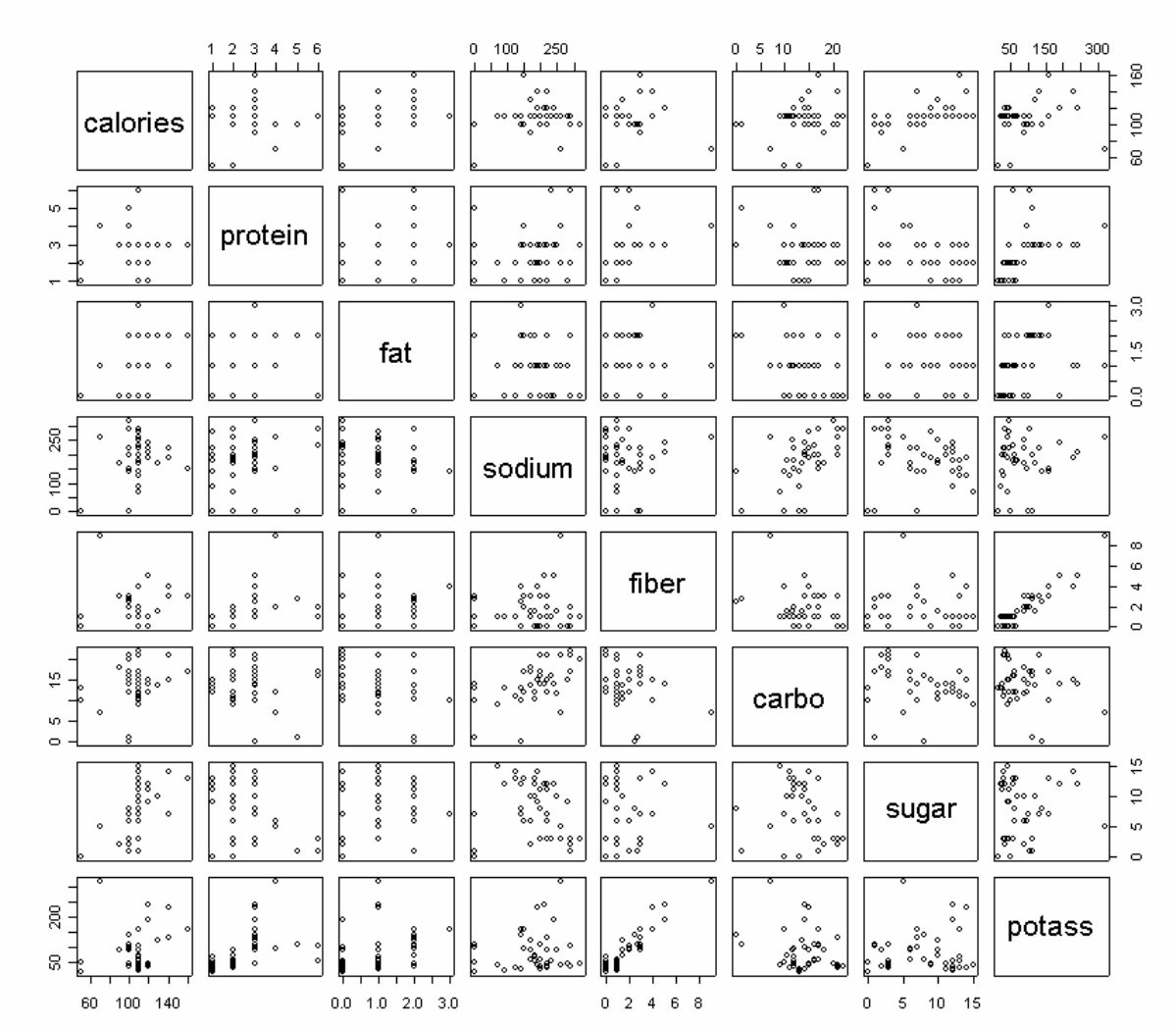
I want to see how the different brands are "positioned" with regard to these characteristics.

If I just think about two of the characteristics at a time this is easy.

How do I see how they are positioned using all 8 ?



Maybe I could drop potass or fiber.



3. Distance

We think of each brand of cereal as an object.

We wish to group together objects which are similar, or equivalently, place objects which are different in different groups.

To do this, we have to have a measure of how different objects are.

In R the difference between two objects is called the distance.

Suppose x denotes the set of measurements for one object and y those of another,
(just to make a point, y does not mean what it has in directed mining!!!!).

For example, with the cereal data and the first two brands:

```
> x = cereal[1,]
> y = cereal[2,]
> x
          calories protein fat sodium fiber carbo sugar potass
ACCheerios      110       2   2    180   1.5  10.5    10      70
> y
         calories protein fat sodium fiber carbo sugar potass
Cheerios      110       6   2    290     2    17     1    105
```

(ACCheerios is Apple-Cinnamon Cheerios)

So, object x is ACCheerios and object y is Cheerios.

How different are objects x and y ?

The first thing you might think of to use is Euclidean distance:

$$d(x,y) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

```
> sqrt(sum((x-y)^2))
[1] 116.0366
```

R has a function to compute distances, since it plays a basic role in clustering (and mds).

```
> library(mva)
> ?dist

Usage:

     dist(x, method = "euclidean", diag = FALSE, upper = FALSE)

     print.dist(x, diag = NULL, upper = NULL, ...)
     as.matrix.dist(x)
.....
```

Here, x would be a data frame, and the function will compute the distances for all pairs of objects, for all rows of the frame.

If we try dist on the first two brands we see that it returns the Euclidean distance:

```
> cer2 = cereal[1:2,]
> cer2
            calories protein fat sodium fiber carbo sugar potass
ACCheerios      110       2   2    180   1.5  10.5    10      70
Cheerios        110       6   2    290   2.0  17.0     1     105
> dist(cer2)
[1] 116.0366
```

The first four objects (brands):

```
> dist(cereal[1:4,])
             ACCheerios    Cheerios       CocoaPuffs
Cheerios      116.036632
CocoaPuffs    15.508062    121.6511
CountChocula   6.363961    117.8940         10
```

So, between each pair of objects we have a distance.

With all 43 brands we have a distance matrix.

```
> temp = as.matrix(dist(cereal))
> dim(temp)
[1] 43 43
> temp[1,1]
[1] 0
> temp[1,2]
[1] 116.0366
> temp[1,3]
[1] 15.50806
> temp[3,1]
[1] 15.50806
```

To access specific distances I turn it into a matrix.

The matrix is 43 by 43 each row and column corresponds to an object (brand).

The distance between object i and object j is temp[i,j].

Object 1 is not far from itself.

Dist between 1 and 2 is as before.

Dist between 1 and 3 is as before.

Dist between 1 and 3 is same as 3 and 1.

Is Euclidean distance a sensible thing to use
for our cereal example ?

Some quantities are measured in grams and others
in milligrams.

21

The dist function in R has 5 options for the form of the distance:

euclidean:

Usual square distance between the two vectors (2 norm).

maximum:

Maximum distance between two components of *x* and *y* (supremum norm)

manhattan:

Absolute distance between the two vectors (1 norm).

canberra:

sum(|x_i - y_i| / |x_i + y_i|). Terms with zero numerator and denominator are omitted from the sum and treated as if the values were missing.

binary:

(aka *asymmetric binary*): The vectors are regarded as binary bits, so non-zero elements are `on' and zero elements are `off'. The distance is the *proportion* of bits in which only one is on amongst those in which at least one is on.

```
> tempdf = data.frame(x=c(1,4),y=c(2,6))
> row.names(tempdf) = c("o1","o2")
> tempdf
   x y
o1 1 2
o2 4 6
> dist(tempdf,method="euclidean")
[1] 5
> dist(tempdf,method="maximum")
[1] 4
> dist(tempdf,method="manhattan")
[1] 7
> dist(tempdf,method="canberra")
[1] 1.1
> (3/5) + (4/8)
[1] 1.1
```

A simple example with 4 different distances applied.

Note that canberra does not depend on the units of the variable.

How about the binary distance?

This is appropriate when each of the variables in binary (factor with two levels).

For example, we might ask consumers a series of yes/no questions about the brands.

Of course, you can always take any variable and “bin” it to make it binary.

The distance is:

times only one var =1 / times at least one is 1

```
> o1 = c(0,0,0,0,1,1,1,1,1)
> o2 = c(0,0,1,1,0,1,1,1,1)
> temp = rbind(o1,o2)
> dist(temp,method="binary")
[1] 0.4285714
> 3/7
[1] 0.4285714
```

At least one of the variables is 1 7 times.

3 of those times only 1 of the two is 1.

dist is 3/7.

For example, if they matched perfectly, the distance would be 0.

25

There are only three possibilities:

both 0, both 1, mismatch.

dist = # mismatch / (# both 1 + # mismatch)

It is not clear that this is the best thing to do !!

Why should these two simple examples give different distances ?

```
> o1 = c(0,0,0,0,0,1)
> o2 = c(0,0,0,0,0,0)
> temp = rbind(o1,o2)
> dist(temp,method="binary")
[1] 1
>
> o1 = c(1,1,1,1,1,0)
> o2 = c(1,1,1,1,1,1)
> temp = rbind(o1,o2)
> dist(temp,method="binary")
[1] 0.1666667
> 1/6
[1] 0.1666667
```

People have looked at just about every possible way you can combine # both 0, # both 1, # mismatch to get a distance measure.

27

In general you should define your own distance measure. This will give a p by p matrix of distances for any pair.

R has functions for turning matrices into distance data structures and vice versa:

```
as.matrix.dist(x)
as.dist(m, diag = FALSE, upper = FALSE)
```

You can also just use as.matrix (as I did, seems to give the same thing).

```
> temp = dist(cereal[1:4,])
> temp
             ACCheerios Cheerios CocoaPuffs
Cheerios     116.036632
CocoaPuffs    15.508062 121.6511
CountChocula   6.363961 117.8940         10
> junk = as.matrix.dist(temp)
> junk
             ACCheerios Cheerios CocoaPuffs CountChocula
ACCheerios     0.000000 116.0366   15.50806     6.363961
Cheerios     116.036632   0.0000  121.65114   117.894020
CocoaPuffs    15.508062 121.6511    0.00000    10.000000
CountChocula   6.363961 117.8940   10.00000     0.000000
> stuff = as.dist(junk)
> stuff
             ACCheerios Cheerios CocoaPuffs
Cheerios     116.036632
CocoaPuffs    15.508062 121.6511
CountChocula   6.363961 117.8940         10
```

Choosing a good distance could be quite difficult to do sensibly in practice.

You could have several different numeric variables with completely different units.

What is the relative size of the distances?

$$d(x,y) = \sqrt{\sum_{i=1}^{p} w_i (x_i - y_i)^2}$$

Choosing the weight is equivalent to choosing a scaling for the variable.

30

How about different factor variables with different numbers of levels combined with numerics having different units ?

31

How about our cereal example ?

Hey this should be easy, we just have grams and milligrams.

Should we convert the grams into milligrams ?

32

Note:

The R documentation calls the distances “dissimilarities”.

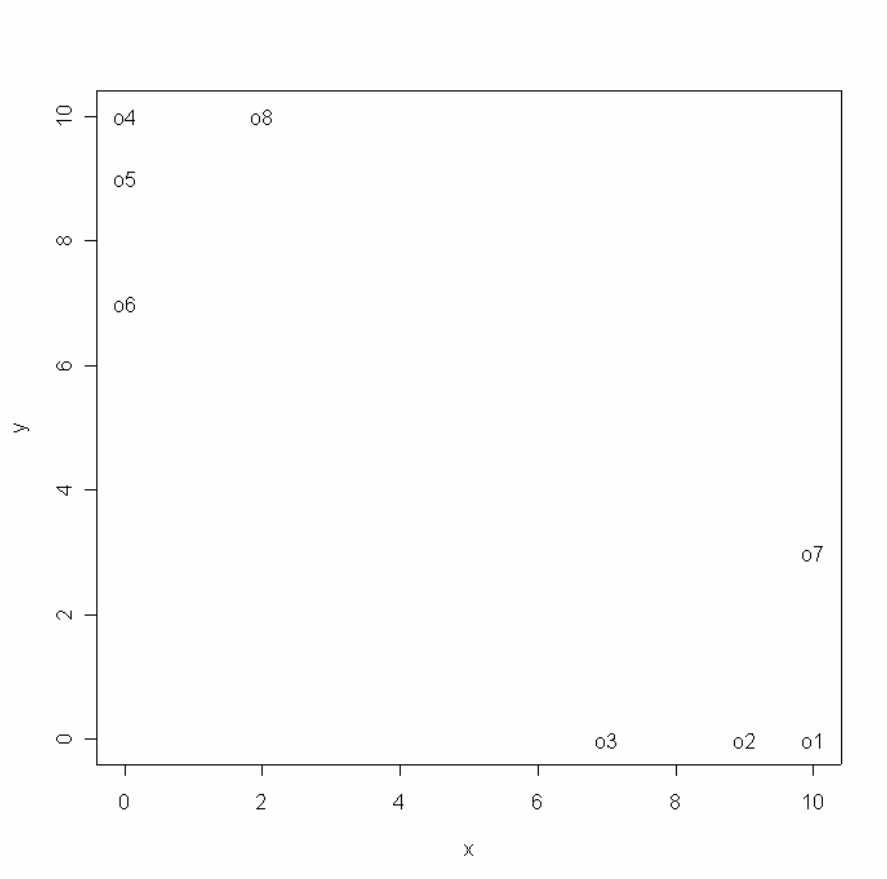
4. Hierarchical Clustering

Lets look at a simple example.

oo is a data frame holding 8 objects.
Each object is represented by two numeric measurements called x and y.

```
> oo
x y
o1 10 0
o2 9  0
o3 7  0
o4 0 10
o5 0  9
o6 0  7
o7 10 3
o8 2 10
```

There
are two
obvious groups.



35

The function in R for hierchical clustering is "hclust".

You have to give it a distance structure.

Here the results are in temp. The merge component tells how the clustering is done.

```
> temp = hclust(dist(oo))
> names(temp)
[1] "merge"       "height"      "order"
[4] "labels"      "method"      "call"
[7] "dist.method"
> temp$merge
     [,1] [,2]
[1,]   -1   -2
[2,]   -4   -5
[3,]   -8    2
[4,]   -3    1
[5,]   -6    3
[6,]   -7    4
[7,]    5    6
```

Each row refers to a step in the clustering procedure. We will go through it line by line, step by step.

Hierarchical clustering starts by thinking of each object as a cluster all by itself.

Then it picks two “clusters” to merge together.

The first row of the temp$merge says the first two objects joined together are o1 and o2.

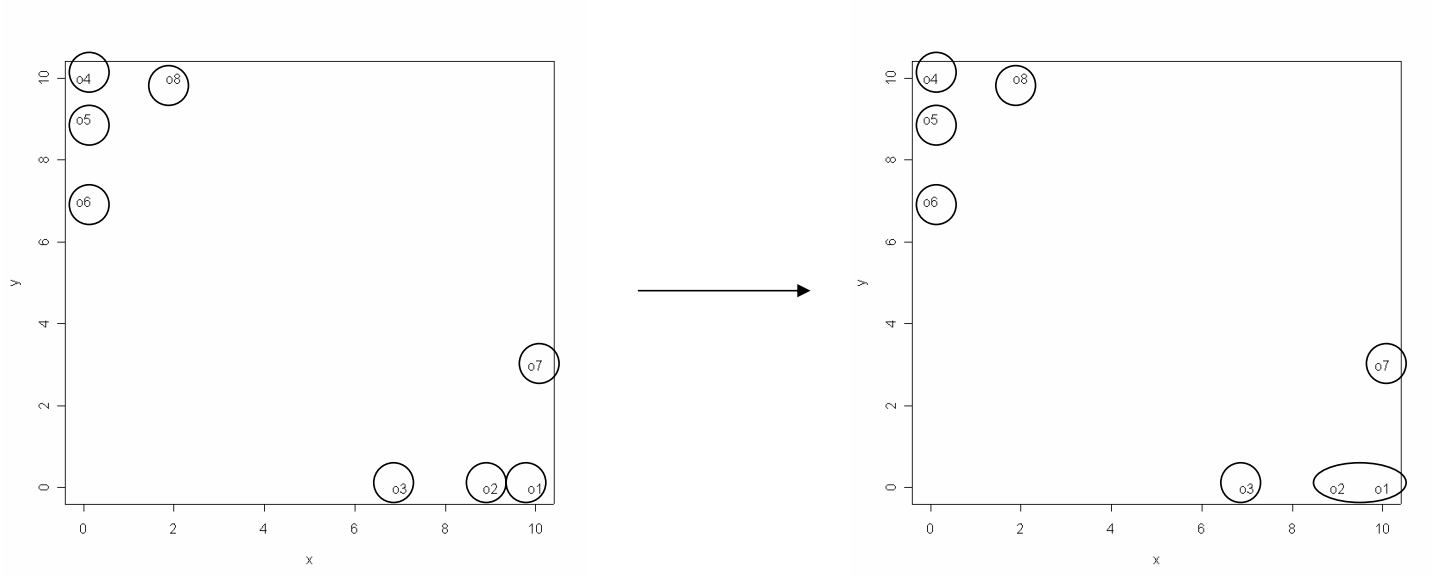
The numbers with a minus sign refers to the actual objects. So -1 refers to the first object.

```
> temp$merge
     [,1] [,2]
[1,]   -1   -2
```

Take objects 1 and 2 and merge them together.

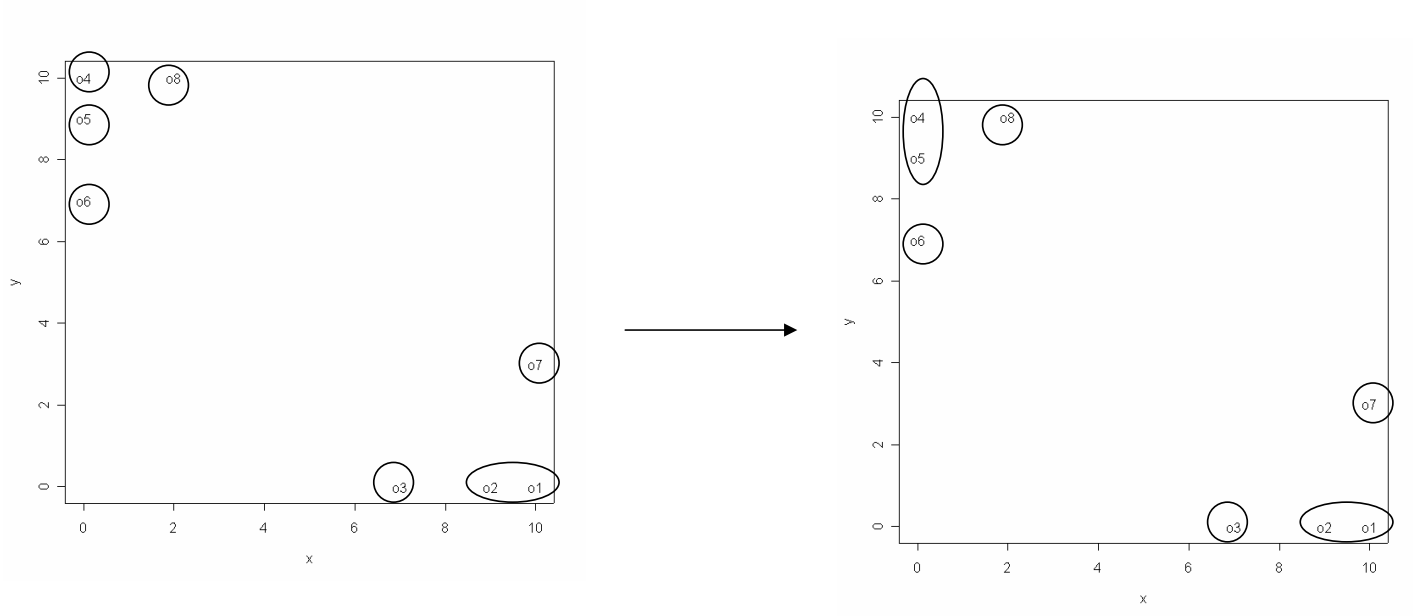
So this is step 1:

```
> temp$merge
     [,1] [,2]
[1,]   -1   -2
```



The next step is to put objects 4 and 5 together.

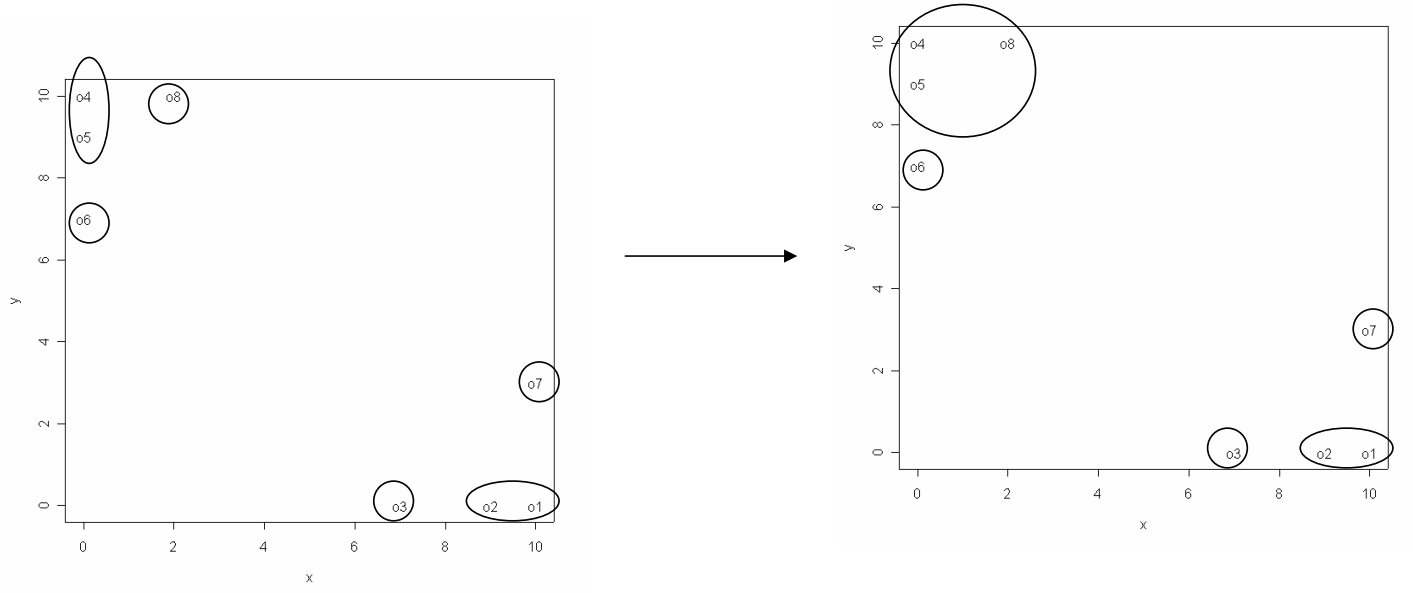
```
> temp$merge
     [,1] [,2]
[1,]   -1   -2
[2,]   -4   -5
```



The next step is to put object 8 and the cluster formed at step 2 together.

```
> temp$merge
     [,1] [,2]
[1,]   -1   -2
[2,]   -4   -5
[3,]   -8    2
```

So, this two refers to the cluster formed at step 2 which was objects 4 and 5 combined.

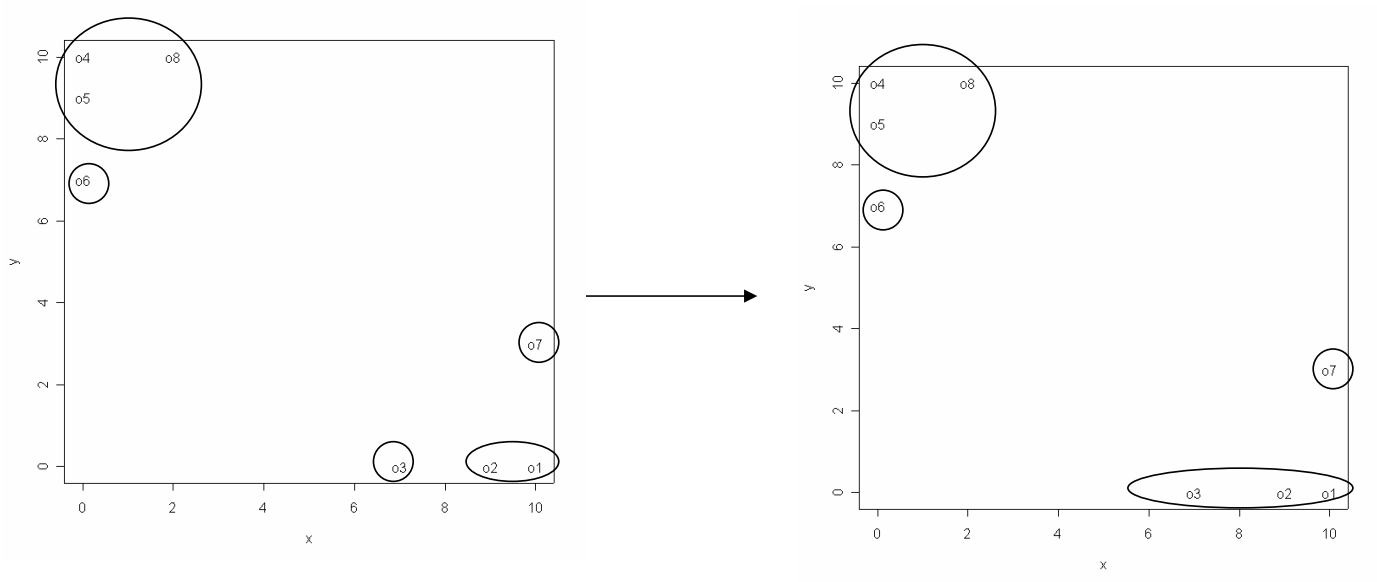


[1,] -1 -2

[2,] -4 -5

[3,] -8 2

[4,] -3 1



41

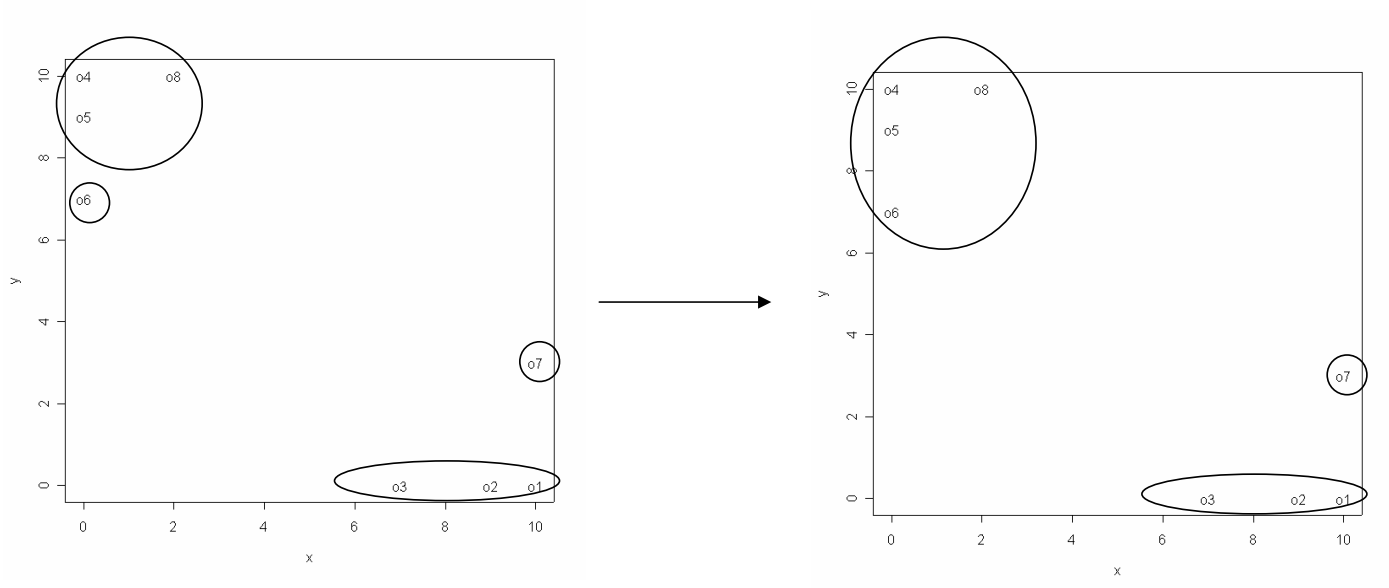
[1,] -1 -2

[2,] -4 -5

[3,] -8 2

[4,] -3 1

[5,] -6 3



1,] -1 -2

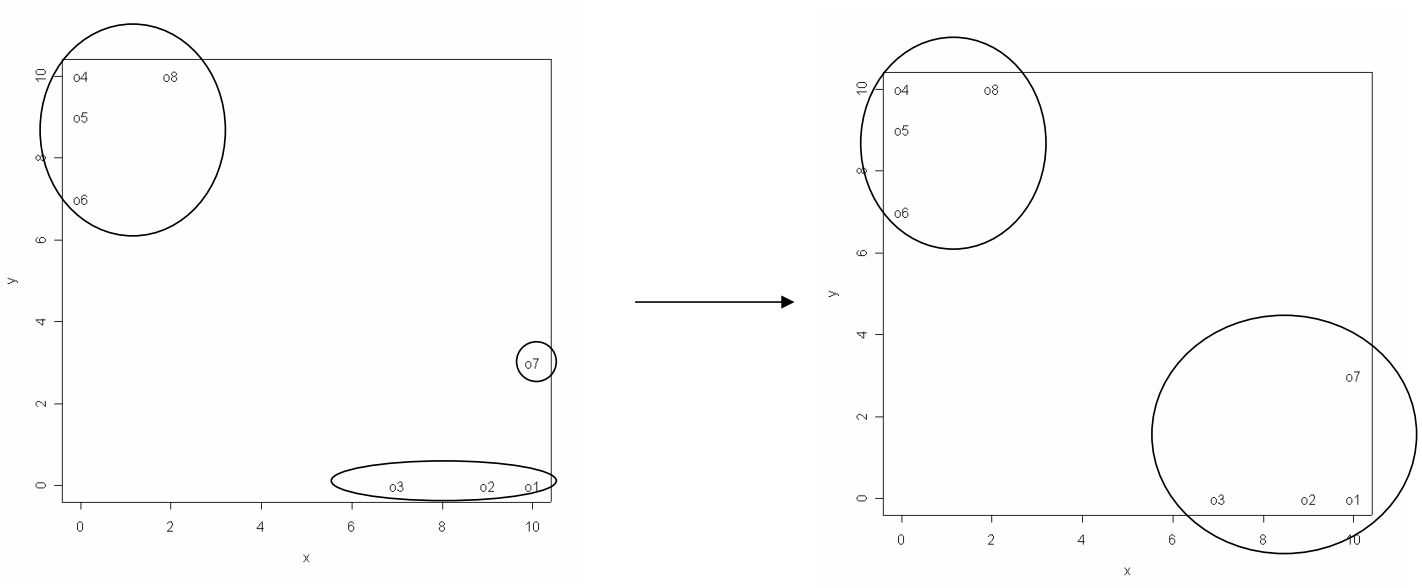
[2,] -4 -5

[3,] -8 2

[4,] -3 1

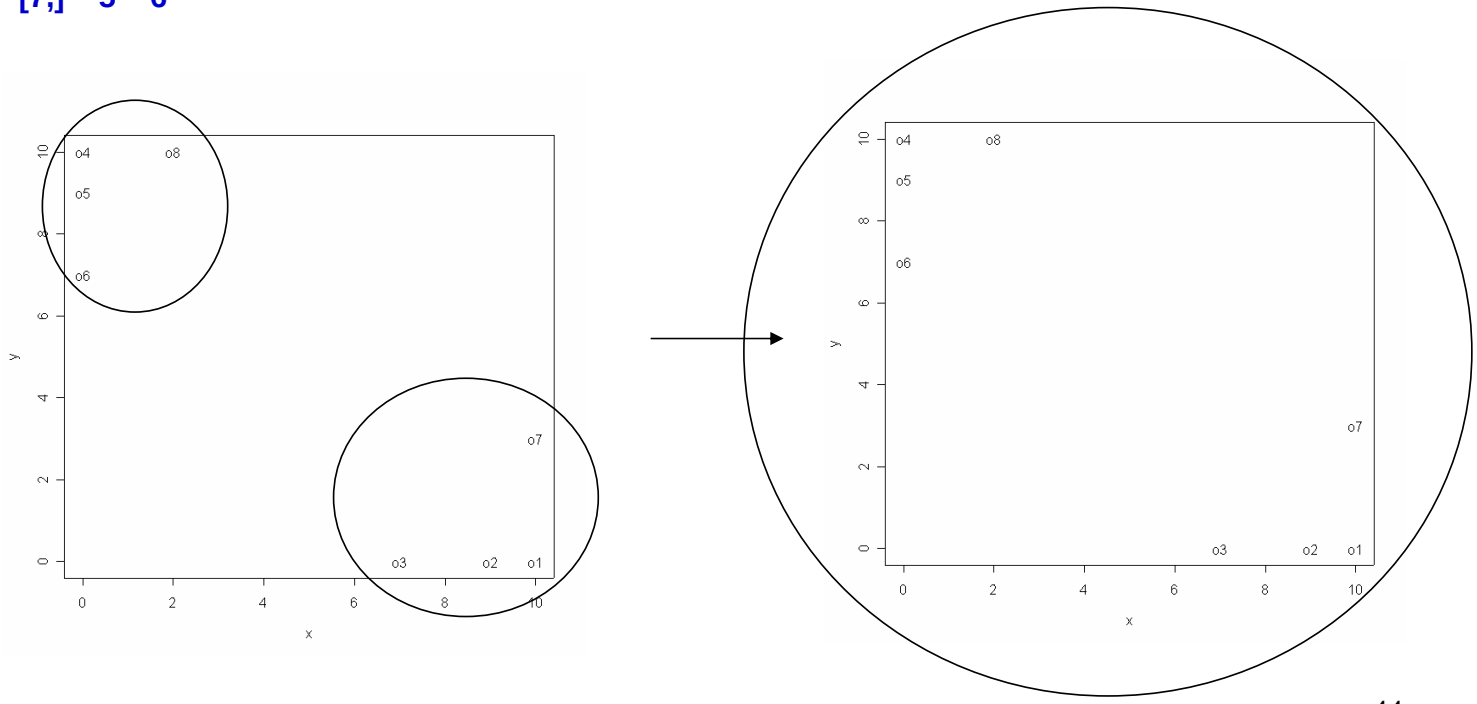
[5,] -6 3

[6,] -7 4



[1,] -1 -2
[2,] -4 -5
[3,] -8 2
[4,] -3 1
[5,] -6 3
[6,] -7 4
[7,] 5 6

Finally, we are all one big happy family.

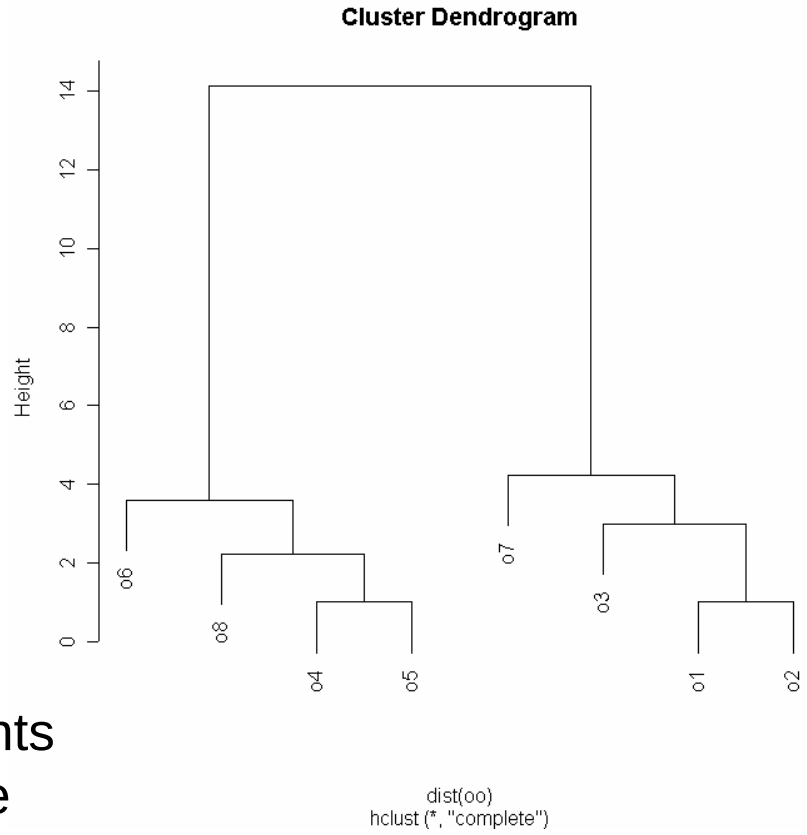


44

How does
R plot the
clustering ?

> plot(temp)

[1,] -1 -2
[2,] -4 -5
[3,] -8 2
[4,] -3 1
[5,] -6 3
[6,] -7 4
[7,] 5 6



The “height” represents the “closeness” of the joined clusters.

Hierarchical Clustering

1.
Start with each object as a cluster by itself.

2.

At each step, combine the two clusters which are closest.

3.

Stop when all objects are combined together.

How do you define how close two *clusters* are ?

“complete” uses the maximum of all distances between pairs of objects formed by selecting one object from each of the two clusters.

“single” uses the minimum.

```
> hclust(dist(oo),method="complete")$merge
     [,1] [,2]
[1,]   -1   -2
[2,]   -4   -5
[3,]   -8    2
[4,]   -3    1
[5,]   -6    3
[6,]   -7    4
[7,]    5    6
> hclust(dist(oo),method="single")$merge
     [,1] [,2]
[1,]   -1   -2
[2,]   -4   -5
[3,]   -3    1
[4,]   -6    2
[5,]   -8    4
[6,]   -7    3
[7,]    5    6
```

47

“complete” will give you “tight” clusters.

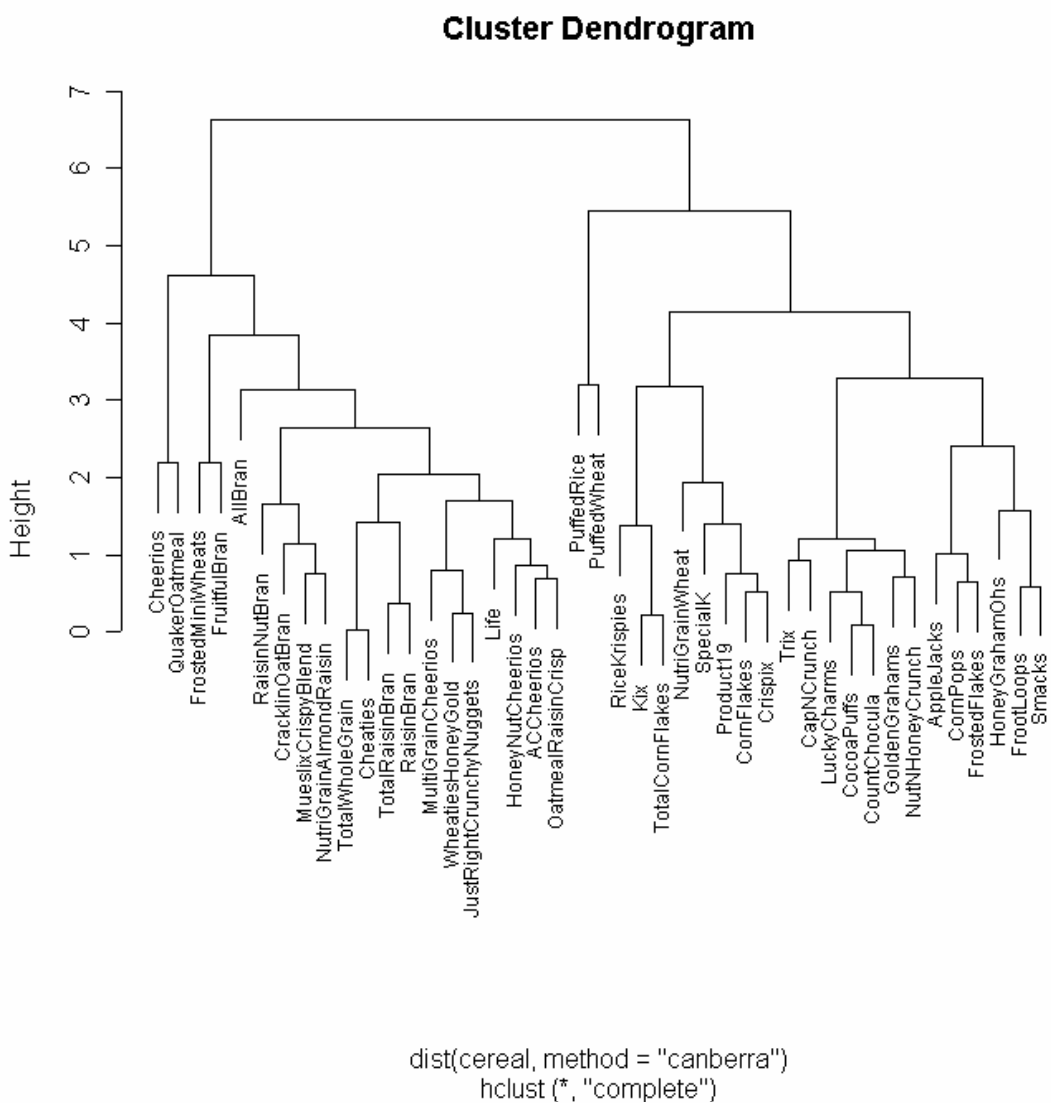
“single” will give you snaky ones.
(any friend of yours is a friend of mine).

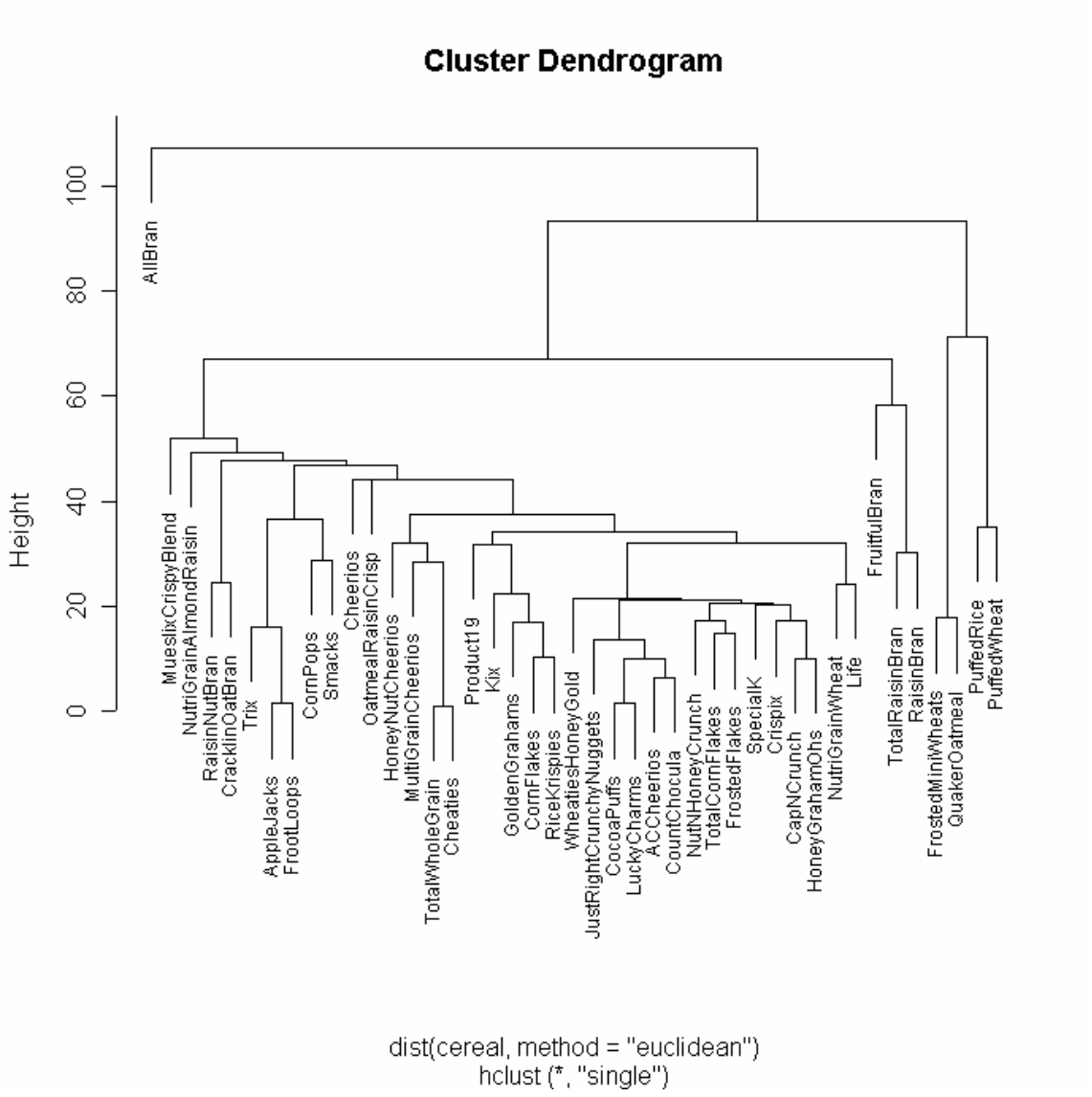
There are several choices.

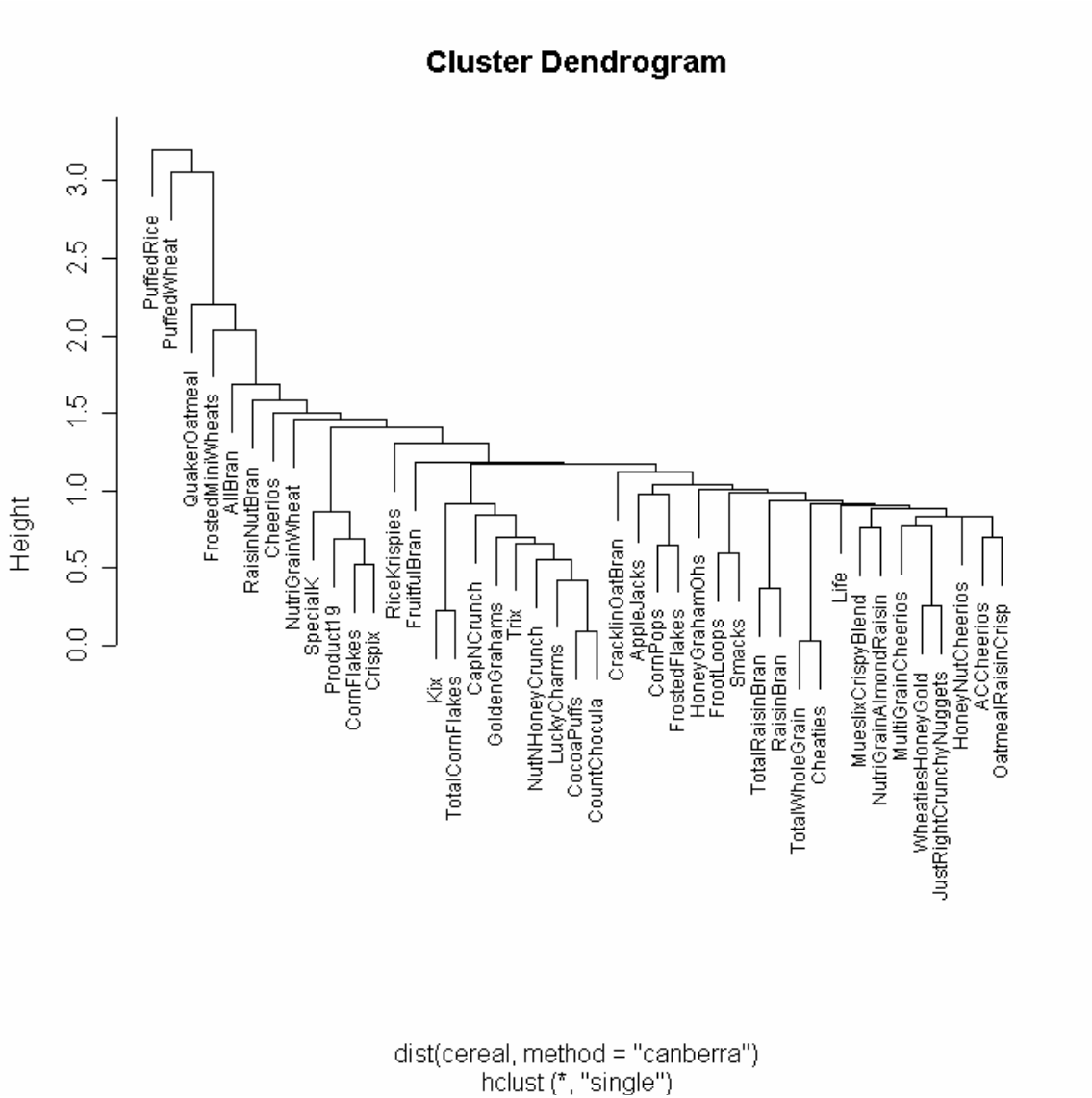
First you have to choose the distance between pairs of objects, then you have to choose how to combine such distances to give the distance between two clusters of objects.

Let’s try the cereal data

```
> cerhc = hclust(dist(cereal,method="canberra"),method="complete")
> names(cerhc)
[1] "merge"       "height"      "order"       "labels"
[5] "method"      "call"        "dist.method"
> plot(cerhc,cex=.75)
```


Cluster Dendrogram
Height
0
1
2
3
4
5
6
7
Cheerios
QuakerOatmeal
FrostedMiniWheats
FruitfulBran
AllBran
RaisinNutBran
CracklinOatBran
MueslixCrispyBlend
NutriGrainAlmondRaisin
TotalWholeGrain
Cheaties
TotalRaisinBran
RaisinBran
MultiGrainCheerios
WheatiesHoneyGold
JustRightCrunchyNuggets
Life
HoneyNutCheerios
ACCheerios
OatmealRaisinCrisp
PuffedRice
PuffedWheat
RiceKrispies
Kix
TotalCornFlakes
NutriGrainWheat
SpecialK
Product19
CornFlakes
Crispix
Trix
CapNCrunch
LuckyCharms
CocoaPuffs
CountChocula
GoldenGrahams
NutNHoneyCrunch
AppleJacks
CornPops
FrostedFlakes
HoneyGrahamOhs
FrootLoops
Smacks
dist(cereal, method = "canberra")
hclust (*, "complete")


Cluster Dendrogram
Height
0
20
40
60
80
100
AllBran
MueslixCrispyBlend
NutriGrainAlmondRaisin
RaisinNutBran
CracklinOatBran
Trix
AppleJacks
FrootLoops
CornPops
Smacks
Cheerios
OatmealRaisinCrisp
HoneyNutCheerios
MultiGrainCheerios
TotalWholeGrain
Cheaties
Product19
Kix
GoldenGrahams
CornFlakes
RiceKrispies
WheatiesHoneyGold
JustRightCrunchyNuggets
CocoaPuffs
LuckyCharms
ACCheerios
CountChocula
NutNHoneyCrunch
TotalCornFlakes
FrostedFlakes
SpecialK
Crispix
CapNCrunch
HoneyGrahamOhs
NutriGrainWheat
Life
FruitfulBran
TotalRaisinBran
RaisinBran
FrostedMiniWheats
QuakerOatmeal
PuffedRice
PuffedWheat
dist(cereal, method = "euclidean")
hclust (*, "single")


Cluster Dendrogram
Height
0.0
0.5
1.0
1.5
2.0
2.5
3.0
PuffedRice
PuffedWheat
QuakerOatmeal
FrostedMiniWheats
AllBran
RaisinNutBran
Cheerios
NutriGrainWheat
SpecialK
Product19
CornFlakes
Crispix
RiceKrispies
FruitfulBran
Kix
TotalCornFlakes
CapNCrunch
GoldenGrahams
Trix
NutNHoneyCrunch
LuckyCharms
CocoaPuffs
CountChocula
CracklinOatBran
AppleJacks
CornPops
FrostedFlakes
HoneyGrahamOhs
FrootLoops
Smacks
TotalRaisinBran
RaisinBran
TotalWholeGrain
Cheaties
Life
MueslixCrispyBlend
NutriGrainAlmondRaisin
MultiGrainCheerios
WheatiesHoneyGold
JustRightCrunchyNuggets
HoneyNutCheerios
ACCheerios
OatmealRaisinCrisp
dist(cereal, method = "canberra")
hclust (*, "single")

52

51

cutree

There are several functions in R for “doing” hierarchical clustering.

At a minimum we need to:

(i) cut the tree at some level to define a set of clusters

(ii) get the cluster id’s of the objects

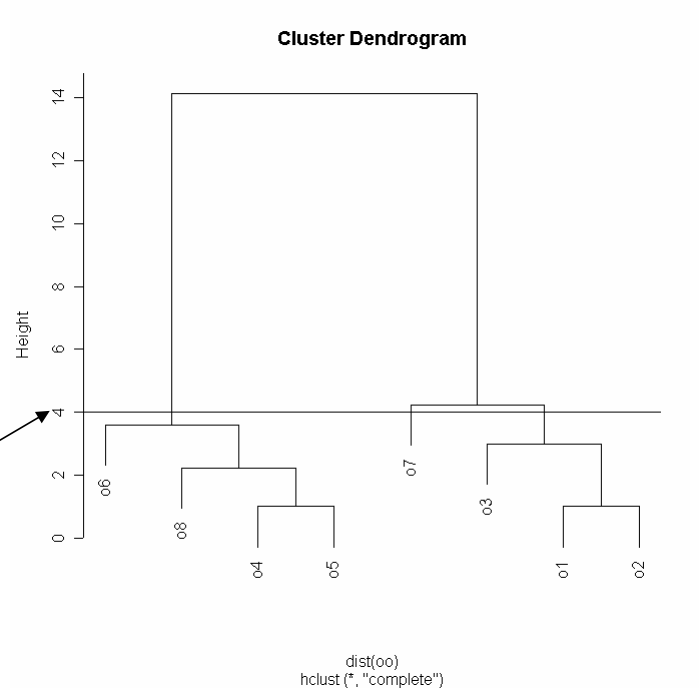
(i) and (ii) are achieved by ***cutree***.

53

```
> oohc = hclust(dist(oo))
> plot(oohc)
> ooc3 = cutree(oohc,3)
> ooc3
o1 o2 o3 o4 o5 o6 o7 o8
 1  1  1  2  2  2  3  2

> cutree(oohc,h=4)
o1 o2 o3 o4 o5 o6 o7 o8
 1  1  1  2  2  2  3  2
```

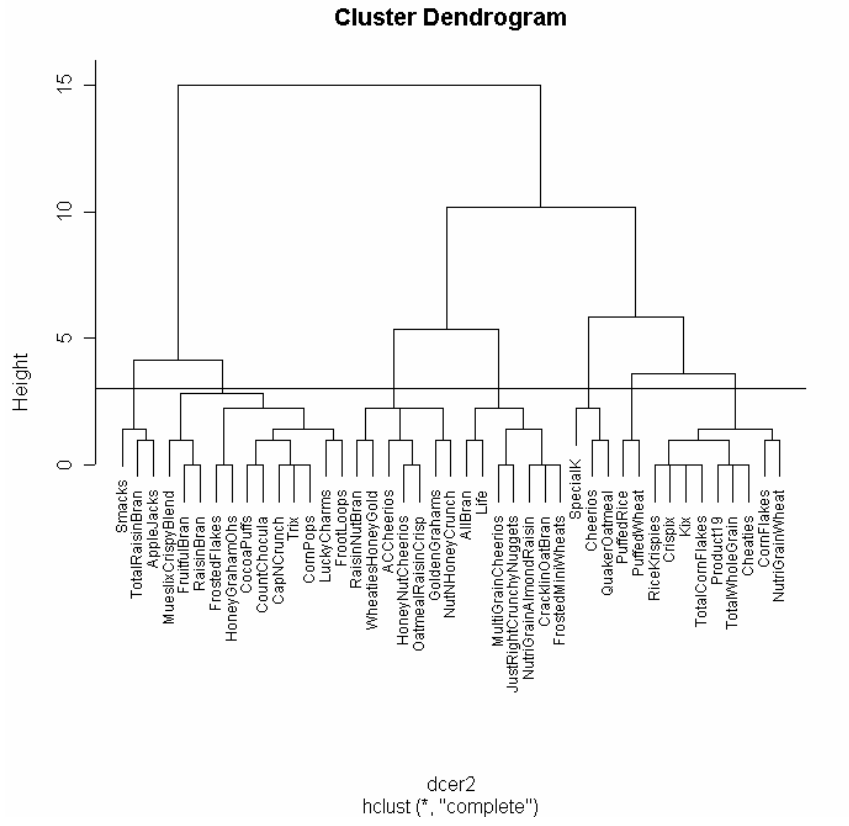
You can ask for a number of groups

or cut the "dendogram" at a certain height.



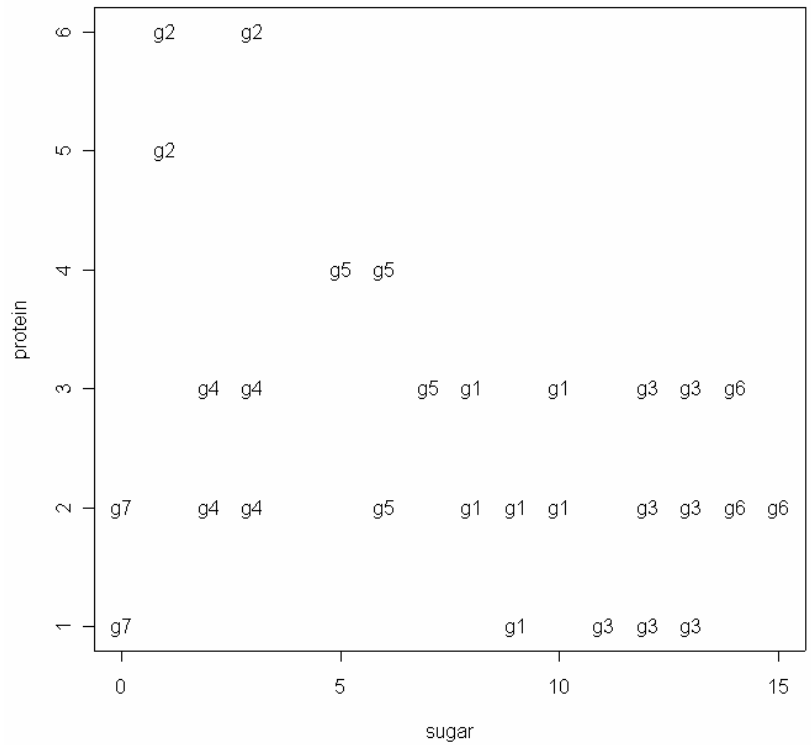
Let’s do cereal with only two variables and then the whole thing.

```
> cer2 = cereal[,c("sugar","protein")]
> dcer2 = dist(cer2,method="euclidean")
> cer2hc = hclust(dcer2)
> plot(cer2hc,cex=.75)
> abline(h=3)
```



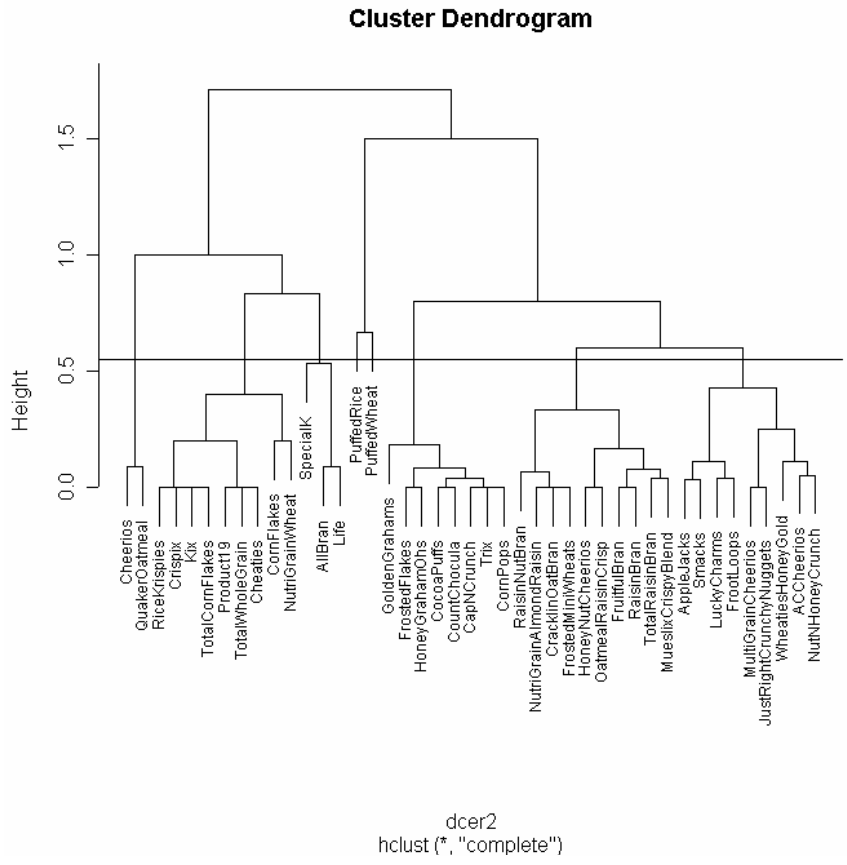
```
> cer2Ind = cutree(cer2hc,h=3)
> table(as.factor(cer2Ind))

 1  2  3  4  5  6  7
 7  3 12  9  7  3  2
>
> plot(cer2,type="n",
xlab="sugar",ylab="protein")
> for(i in 1:length(cer2Ind)){
+ text(cer2$sugar[cer2Ind==i],
cer2$protein[cer2Ind==i],
paste("g",i,sep=""))
+ }
```

Plot of the two variables with the group labels.

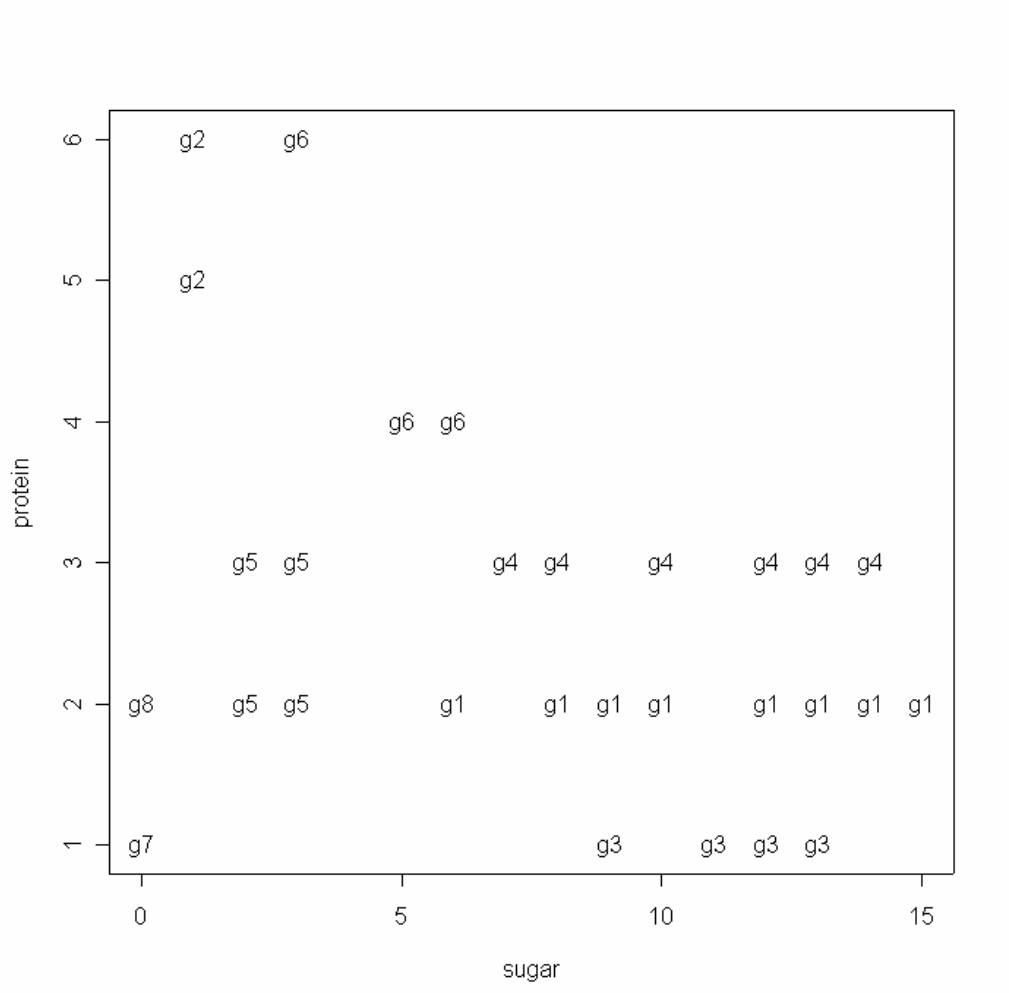


Let’s change it
to “canberra”.

```
> dcer2 = dist(cer2,method="canberra")
> cer2hc = hclust(dcer2)
> plot(cer2hc,cex=.75)
> abline(h=.55)
```



Different.



58

Cereal, the real thing:

cerhc =
hclust(dist(cereal,method="canberra")
,method="complete")

> plot(cerhc,cex=.75)

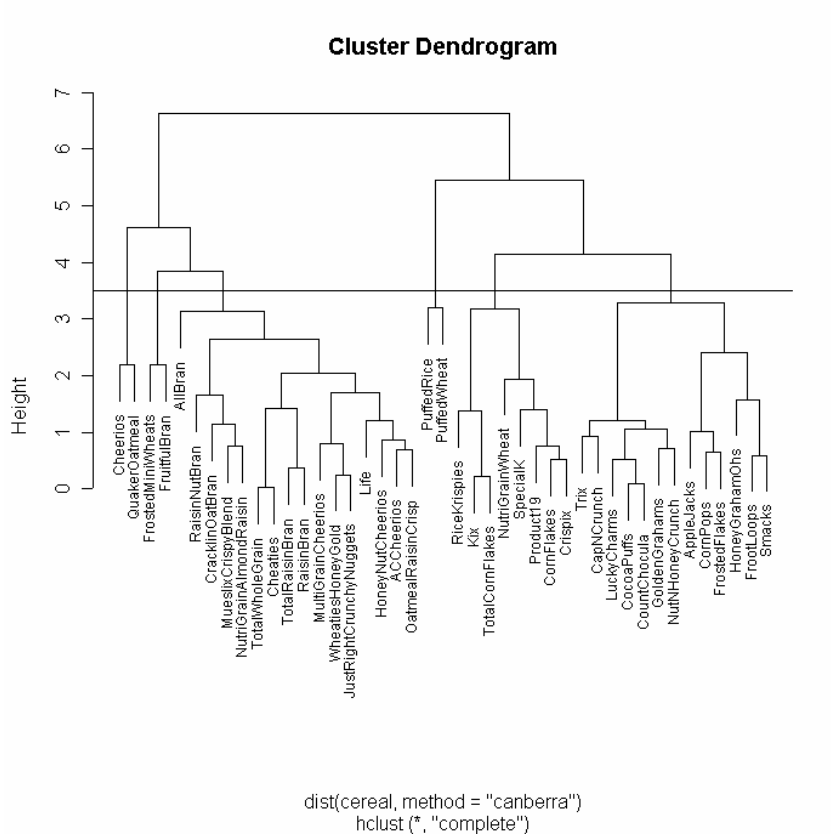
> abline(h=3.5)

> cerInd = cutree(cerhc,h=3.5)

> table(as.factor(cerInd))

1 2 3 4 5 6

16 2 13 8 2 2



```
> row.names(cereal)[cerInd==1]
 [1] "ACCheerios"           "HoneyNutCheerios"
 [3] "MultiGrainCheerios"   "OatmealRaisinCrisp"
 [5] "RaisinNutBran"        "TotalRaisinBran"
 [7] "TotalWholeGrain"      "Cheaties"
 [9] "WheatiesHoneyGold"    "AllBran"
[11] "CracklinOatBran"      "JustRightCrunchyNuggets"
[13] "MueslixCrispyBlend"   "NutriGrainAlmondRaisin"
[15] "RaisinBran"           "Life"
> row.names(cereal)[cerInd==3]
 [1] "CocoaPuffs"     "CountChocula"   "GoldenGrahams"  "LuckyCharms"
 [5] "Trix"           "AppleJacks"     "CornPops"       "FrootLoops"
 [9] "FrostedFlakes"  "NutNHoneyCrunch" "Smacks"        "CapNCrunch"
[13] "HoneyGrahamOhs"
> row.names(cereal)[cerInd==4]
[1] "Kix"           "TotalCornFlakes" "CornFlakes"    "Crispix"
[5] "NutriGrainWheat" "Product19"     "RiceKrispies"  "SpecialK"
> row.names(cereal)[cerInd==2]
[1] "Cheerios"     "QuakerOatmeal"
> row.names(cereal)[cerInd==5]
[1] "FrostedMiniWheats" "FruitfulBran"
> row.names(cereal)[cerInd==6]
[1] "PuffedRice"  "PuffedWheat"
```

How many groups?

Few enough that you can think about them.

But you don't want to combine together things that are really different.

I've probably chosen too few here.

Usually people make up names for the groups.

We can easily obtain the variable summaries for a cluster.

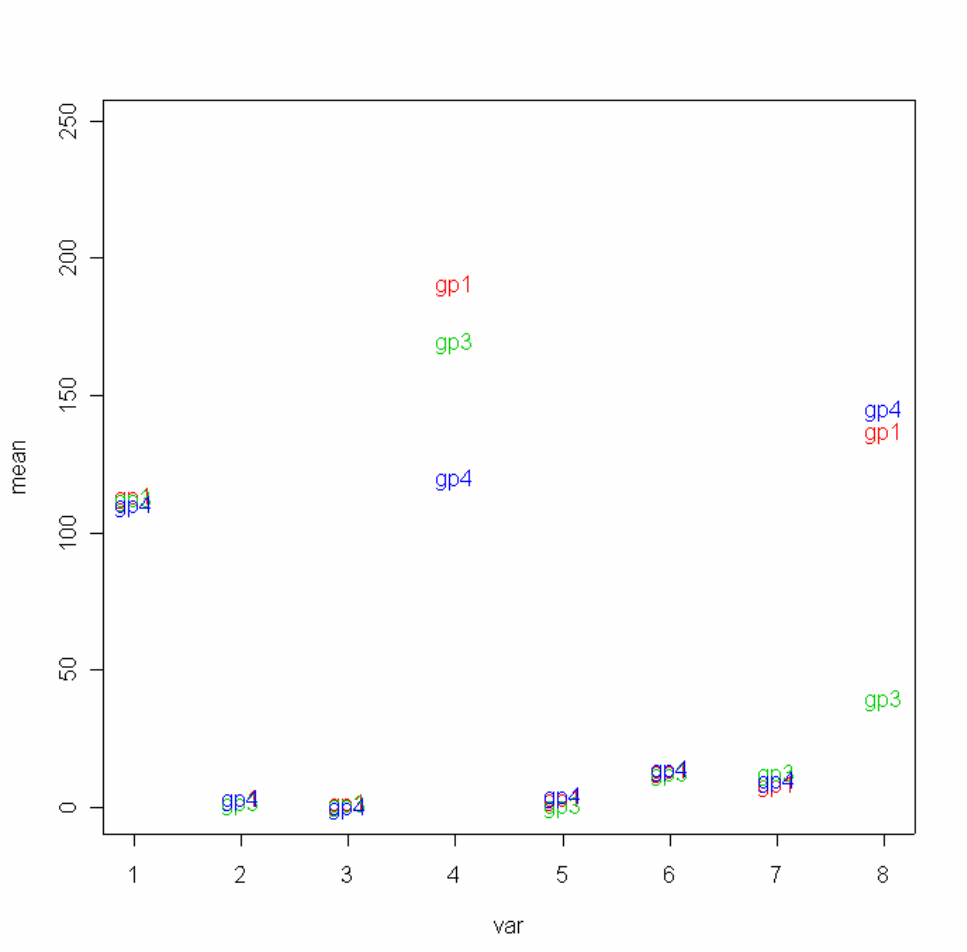
```
> summary(cereal[cerInd==1,])
    calories        protein           fat            sodium      
 Min.   : 70.0   Min.   :2.000   Min.   :1.0   Min.   :140.0  
 1st Qu.:100.0   1st Qu.:2.750   1st Qu.:1.0   1st Qu.:165.0  
 Median :110.0   Median :3.000   Median :1.0   Median :195.0  
 Mean   :113.1   Mean   :2.875   Mean   :1.5   Mean   :190.6  
 3rd Qu.:122.5   3rd Qu.:3.000   3rd Qu.:2.0   3rd Qu.:212.5  
 Max.   :160.0   Max.   :4.000   Max.   :3.0   Max.   :260.0  
     fiber            carbo           sugar           potass      
 Min.   :1.000   Min.   : 0.00   Min.   : 3.0   Min.   : 60.0  
 1st Qu.:1.500   1st Qu.:11.25   1st Qu.: 6.0   1st Qu.: 90.0  
 Median :2.750   Median :14.50   Median : 7.5   Median :115.0  
 Mean   :2.938   Mean   :13.28   Mean   : 8.0   Mean   :136.6  
 3rd Qu.:3.250   3rd Qu.:16.25   3rd Qu.:10.0   3rd Qu.:160.0  
 Max.   :9.000   Max.   :21.00   Max.   :14.0   Max.   :320.0
```

I'd like to see how the cluster differ.
I could print summaries for each cluster but
it might not be easy to compare.

I'll plot the cluster means for the different groups.

```
m1 = mean(cereal[cerInd==1,])
m3 = mean(cereal[cerInd==3,])
m4 = mean(cereal[cerInd==4,])
plot(c(1,8),range(c(m1,m3,m4)),xlab="var",ylab="mean",type="n")
text(1:8,m1,"gp1",col=2)
text(1:8,m3,"gp3",col=3)
text(1:8,m5,"gp4",col=4)
```

No use because of the different scales.



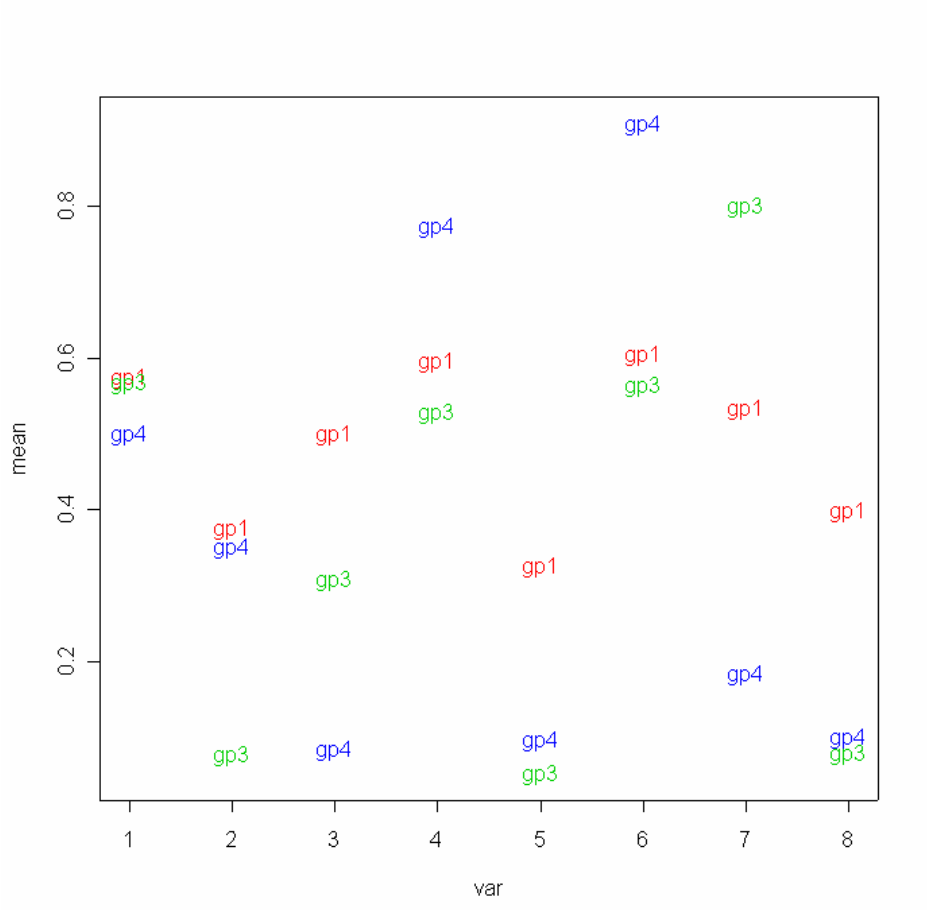
63

I'll scale all the vars to be between 0 and 1, and then plot the means.

```
cerealsc = cereal
for(i in 1:8){
temp = range(cerealsc[[i]])
cerealsc[[i]] = (cerealsc[[i]]-temp[1])/(temp[2]-temp[1])
}

m1 = mean(cerealsc[cerInd==1,])
m3 = mean(cerealsc[cerInd==3,])
m4 = mean(cerealsc[cerInd==4,])
plot(c(1,8),range(c(m1,m3,m4)),xlab="var",ylab="mean",type="n")
text(1:8,m1,"gp1",col=2)
text(1:8,m3,"gp3",col=3)
text(1:8,m4,"gp4",col=4)
```

```
> names(cerealsc)
[1] "calories" "protein"  "fat"      "sodium"   "fiber"
"carbo"
[7] "sugar"    "potass"
```

Group 3 is high
on sugar
and low on
protein,
fiber,
and potassium.



5. K-means Clustering

k-means is another popular clustering method.

For example, it is in h2o.

The Algorithm:

1. Choose the number of clusters.
2. Choose starting values for the mean vector of each cluster.
3. Assign each object to the cluster having the closest mean
4. Replace the old cluster means with the mean of the cluster
5. Repeat 3 and 4 until “done”

Note: by mean vector, I mean a vector of means for each of the variables.

The help.

Description

Perform k-means clustering on a data matrix.

Usage

```
kmeans(x, centers, iter.max = 10)
```

Arguments

x	A numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).
centers	Either the number of clusters or a set of initial cluster centers. If the first, a random set of rows in x are chosen as the initial centers.
iter.max	The maximum number of iterations allowed.

Let’s try it.

I’m using the scaled cereal data.

```
> cerkm = kmeans(cerealsc,5)
> names(cerkm)
[1] "cluster"  "centers"  "withinss" "size"
> cerkm$cluster
 [1] 5 1 5 5 5 5 2 5 2 4 3 2 4 2 5 2 5 3 5 2 5 3 2 5 5 2 4 2 4 5 4 2 2 4 2 5 1 5 5 3 2 2 3
> cerkm$centers
  calories   protein       fat    sodium      fiber     carbo     sugar    potass
1 0.5454545 1.0000000 0.3333333 0.8125000 0.16666667 0.7500000 0.1333333 0.2131148
2 0.4155844 0.2571429 0.1428571 0.5669643 0.15079365 0.7987013 0.2095238 0.1487119
3 0.4181818 0.5600000 0.6666667 0.4312500 0.44888889 0.2727273 0.3600000 0.4918033
4 0.7727273 0.4000000 0.4444444 0.6145833 0.39814815 0.7159091 0.7555556 0.5355191
5 0.5625000 0.1125000 0.3333333 0.5527344 0.06944444 0.5653409 0.7666667 0.1004098
```

So, $cluster give us the cluster id for each object.
$centers gives us the variable means for each cluster.

```
> cerkm$withinss
[1] 0.2693324 3.4848313 1.8653941 0.9931863 1.8572751
> cerkm$size
[1]  2 14  5  6 16
```

$withinss gives us

$$\sum_{i=1}^{n_c}\sum_{j=1}^{p}(x_{ij} - \bar{x}_j)^2$$ n_c = # objects in the cluster

for each cluster.

$size gives us the number of objects in each cluster (n_c).

And that's it !!

71

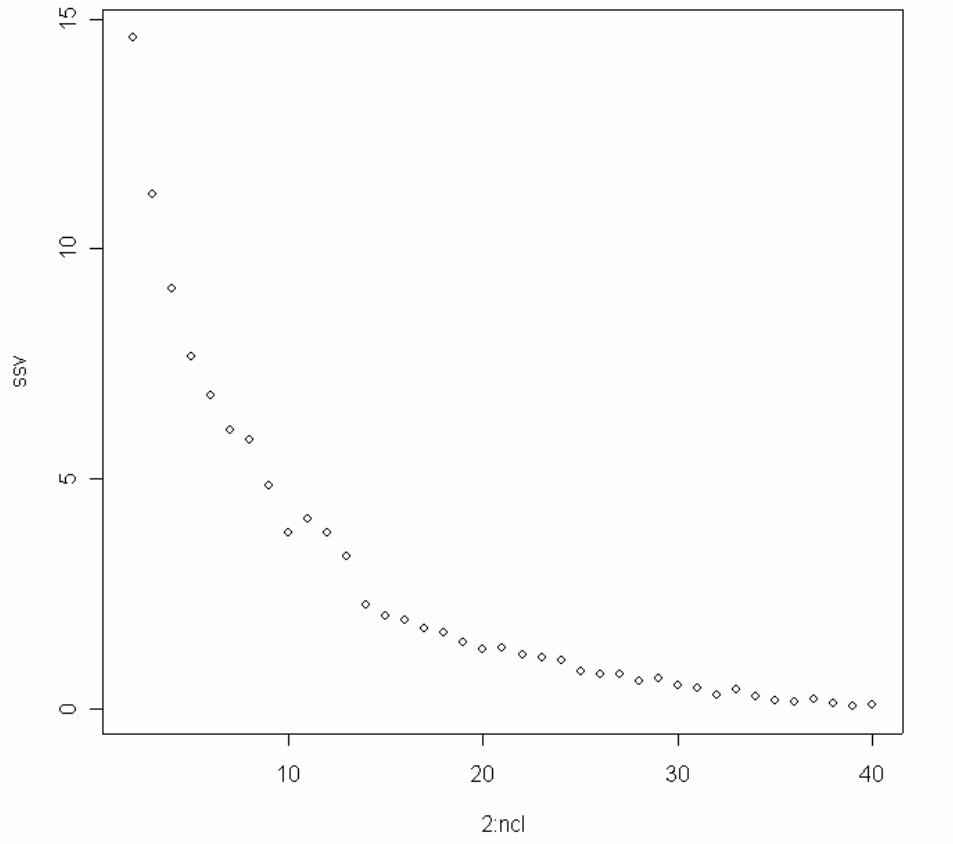
How many clusters?

I'll try various cluster sizes and for each size sum the $withinss.

A "good fit" means this is small.

```
ncl = 40
ssv = rep(0,ncl-1)
for(i in 2:ncl)
{
temp = kmeans(cerealsc,i)
print(i)
print(temp$size)
ssv[i-1] = sum(temp$withinss)
}
plot(2:ncl,ssv)
```

Suggests no more than 13 clusters, but lot's of possibilities.



73

For k-means it seems like we had to make fewer choices than with the hierarchical method.

But don’t be fooled.

Choosing the scale of the data affects the clustering.

74