Displays for Statistics 5401

Lecture 39

December 9, 2005

Christopher Bingham, Instructor
612-625-1024

Class Web Page

`http://www.stat.umn.edu/~kb/classes/5401`

Here is how you might compute

$$\mathbf{B}_p \equiv \sum_{1\le j\le g} p_j(\overline{\mathbf{x}}_j - \overline{\overline{\mathbf{x}}}^{(p)})(\overline{\mathbf{x}}_j - \overline{\overline{\mathbf{x}}}^{(p)})'$$

for the Day-Fisher data with all $p_i = 1/6$.

I started by computing the needed statistics using **groupcovar()**.

```
Cmd> stats <- groupcovar(place,y); stats
component: n              Sample sizes
(1)          5          5          3          3          8          5
component: means          Rows correspond to places
(1,1)    724.2        261      300.4      277.6        704     1046.4
(2,1)    731.8        303      249.2      370.4      804.8     1050.4
(3,1)   688.67        335        238     508.67     1006.3       1090
(4,1)   683.33     388.33         40        525       1063       1018
(5,1)   659.62     217.88        118     368.25     761.25     958.38
(6,1)    658.6        198      131.8      268.2      702.2     1009.8
component: covariance     Pooled variance matrix = E/fe
(1,1)   3300.7     958.53       1440     615.11    -687.86     172.99
(2,1)   958.53     3033.9    -409.91     1610.5     2246.3    -479.11
(3,1)     1440    -409.91     6733.9    -67.496    -340.55     1689.7
(4,1)   615.11     1610.5    -67.496     4720.8     3722.7     886.28
(5,1)  -687.86     2246.3    -340.55     3722.7     5521.2    -159.75
(6,1)   172.99    -479.11     1689.7     886.28    -159.75     1943.9
```

Take transpose so means for each group are in the columns of **means**.

```
Cmd> means <- stats$means' # sample means are now columns

Cmd> spooled <- stats$covariance # pooled variance matrix

Cmd> prior <- rep(1/6,6) # equal prior probabilities

Cmd> ybar_p <- vector(means %*% prior) #weighted sum of cols

Cmd> ybar_p # grand mean
(1)    691.04    283.87    179.57    386.35    840.26    1028.8
```

```
Cmd> d <- means - ybar_p # deviations from grand mean
```

```
Cmd> b_p <- d %*% dmat(prior) %*% d'; b_p
(1,1)   810.87    658.97    1877.6   -202.28   -185.62    705.13
(2,1)   658.97    4357.5   -809.74      5710    8544.5    1381.8
(3,1)   1877.6   -809.74    8069.2   -3282.8   -4809.6    2326.3
(4,1)  -202.28      5710   -3282.8     10092     14054    1208.2
(5,1)  -185.62    8544.5   -4809.6     14054     20387    2130.2
(6,1)   705.13    1381.8    2326.3    1208.2    2130.2    1659.8
```

Find 2 relative eigenvectors of $\mathbf{B}_p$ relative to $\mathbf{S}_{\text{pooled}}$ to get coefficients for $\hat{z}_1$, $\hat{z}_2$:

```
Cmd> eigs <- releigen(b_p,spooled); eigs$values#rel eigenvalues
(1)    5.1663    1.7575   0.57115   0.22092  0.013968 6.4263e-16
```

```
Cmd> u_p <- eigs$vectors[,run(2)] # extract 1st 2 eigenvectors
```

```
Cmd> z <- y %*% u_p # two classification canonical variables
```

Now find the linear discriminant function that uses the columns of z as classifiers.

```
Cmd> discrimfnz <- discrim(place, z); discrimfnz
component: coefs
       place1    place2    place3    place4    place5    place6
(1)    33.502    34.946    37.809    38.882    32.894    33.649
(2)    21.888    21.215    20.766    19.087    17.961    19.847
component: addcon
       place1    place2    place3    place4    place5    place6
(1)   -800.71   -835.67   -930.37   -938.06   -702.31   -763.07
```

Find scores and posterior probabilities.

```
Cmd> d <- z %*% discrimfnz$coefs + discrimfnz$addcon # scores
```

```
Cmd> kx <- d[,1]
```

```
Cmd> post <- exp(d- kx)/sum(exp(d- kx)')' # posterior probs
```

```
Cmd> placez <- vector(grade(post',down:T)[1,]) # Guesses
```

```
Cmd> print(format:"5.3f",hconcat(post,placez,place))
MATRIX:  1     2     3     4     5     6 guessed true
 (1,1) 0.859 0.043 0.000 0.000 0.001 0.097 1.000 1.000
 (2,1) 0.860 0.135 0.000 0.000 0.000 0.005 1.000 1.000
 (3,1) 0.880 0.065 0.000 0.000 0.000 0.055 1.000 1.000
 (4,1) 0.261 0.375 0.001 0.000 0.006 0.358 2.000 1.000 *
 (5,1) 0.274 0.574 0.003 0.000 0.001 0.148 2.000 1.000 *
 (6,1) 0.402 0.549 0.002 0.000 0.000 0.047 2.000 2.000
 (7,1) 0.345 0.635 0.009 0.000 0.000 0.012 2.000 2.000
 (8,1) 0.208 0.428 0.001 0.000 0.006 0.357 2.000 2.000
 (9,1) 0.033 0.644 0.299 0.003 0.000 0.020 2.000 2.000
(10,1) 0.035 0.200 0.002 0.000 0.091 0.672 6.000 2.000 *
(11,1) 0.017 0.451 0.522 0.005 0.000 0.006 3.000 3.000
(12,1) 0.000 0.030 0.905 0.065 0.000 0.000 3.000 3.000
(13,1) 0.000 0.000 0.193 0.807 0.000 0.000 4.000 3.000 *
(14,1) 0.000 0.000 0.116 0.884 0.000 0.000 4.000 4.000
(15,1) 0.000 0.000 0.173 0.827 0.000 0.000 4.000 4.000
(16,1) 0.000 0.000 0.086 0.914 0.000 0.000 4.000 4.000
(17,1) 0.002 0.000 0.000 0.000 0.857 0.141 5.000 5.000
(18,1) 0.010 0.016 0.000 0.000 0.469 0.505 6.000 5.000 *
(19,1) 0.007 0.002 0.000 0.000 0.680 0.311 5.000 5.000
(20,1) 0.000 0.000 0.000 0.000 0.971 0.029 5.000 5.000
(21,1) 0.000 0.000 0.000 0.000 0.981 0.019 5.000 5.000
(22,1) 0.000 0.000 0.000 0.000 0.997 0.003 5.000 5.000
(23,1) 0.027 0.202 0.003 0.000 0.101 0.667 6.000 5.000 *
(24,1) 0.000 0.000 0.000 0.000 0.934 0.065 5.000 5.000
(25,1) 0.384 0.116 0.000 0.000 0.019 0.481 6.000 6.000
(26,1) 0.067 0.044 0.000 0.000 0.190 0.700 6.000 6.000
(27,1) 0.184 0.438 0.002 0.000 0.007 0.369 2.000 6.000 *
(28,1) 0.012 0.005 0.000 0.000 0.574 0.409 5.000 6.000
(29,1) 0.014 0.338 0.036 0.005 0.064 0.542 6.000 6.000
```

```
Cmd> N <- nrows(place)
```

```
Cmd> sum(placez != place)/N
(1,1)   0.27586          APER
```

There are 7 errors, a worse APER (and worse estimated TPM) than using all the variables.

Of course, this is is the result of applying $\hat{\hat{\pi}}$ to the training sample, which we know gives biased estimates of TPM.

Using **jackknife()** with the canonical variables as data ought to be better:

```
Cmd> probs <- jackknife(place, z)

Cmd> sum(place != probs[,7])/N
(1,1)   0.44828      APER_JK
```

But this isn't really doing the leave-one-out thing, since the canonical variables are computed from all the cases.

To do it right, you need to compute different canonical variables for each case. It could be done by brute force, but I didn't try.

discrimfnz$coefs are coefficients for z_1 and z_2. For actual use you would want coefficients that apply directly to **x**.

The 6 vectors (one for each place) of coefficients to multiply x_1, x_2, ..., x_6 are linear combinations of the first two relative eigenvectors, weighted with the columns of **discrimfnz$coefs**. The additive constants are the same.

```
Cmd> coeffs <- u_p %*% discrimfnz$coefs; coeffs
        place1    place2    place3    place4    place5    place6
(1)    0.37894   0.39108   0.41743   0.42337   0.36289   0.37505
(2)     0.1572   0.14827   0.13915   0.12106   0.12004   0.13712
(3)   -0.14449  -0.15786  -0.18046   -0.1956  -0.15747  -0.15456
(4)   -0.68642  -0.69531  -0.72424  -0.71573  -0.62874  -0.66206
(5)    0.76134   0.79022   0.84962   0.86818   0.73891   0.75946
(6)    0.94031   0.95309   0.99358   0.98282   0.86261   0.90773

Cmd> d[run(2),]# scores computed above from canonical variables
        place1    place2    place3    place4    place5    place6
(1)     759.15    756.16    744.07    734.36    752.16    756.97
(2)     845.56     843.7    835.29    824.53    831.35    840.34

Cmd> y[run(2),] %*% coeffs + discrimfnz$addcon # from data
        place1    place2    place3    place4    place5    place6
(1)     759.15    756.16    744.07    734.36    752.16    756.97
(2)     845.56     843.7    835.29    824.53    831.35    840.34
```

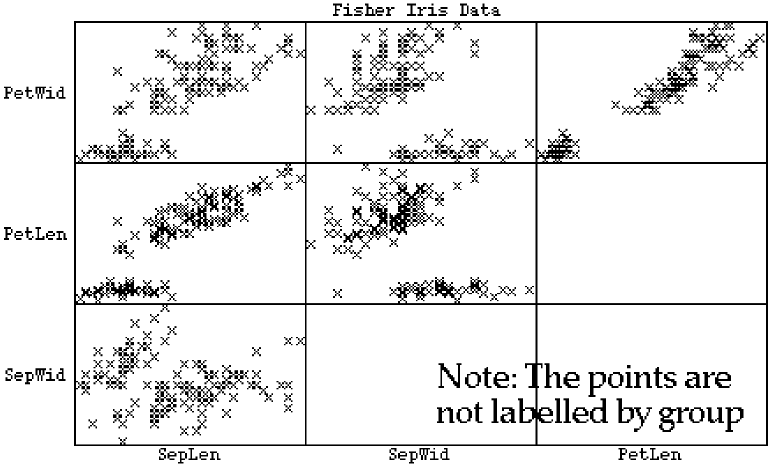
The scores are the same whether computed from **z** or using **coeffs** to compute linear combinations of columns of **y**.

Statement of the clustering problem

Data: Information on N "objects" O_1, ..., O_N that determines

- d_{ij}, a measure of how different or dissimilar are O_i and O_j, *or*
- s_{ij}, a measure of how similar they are.

Goal: Group the objects into a "small" number of "clusters" -- groupings of "similar" objects.



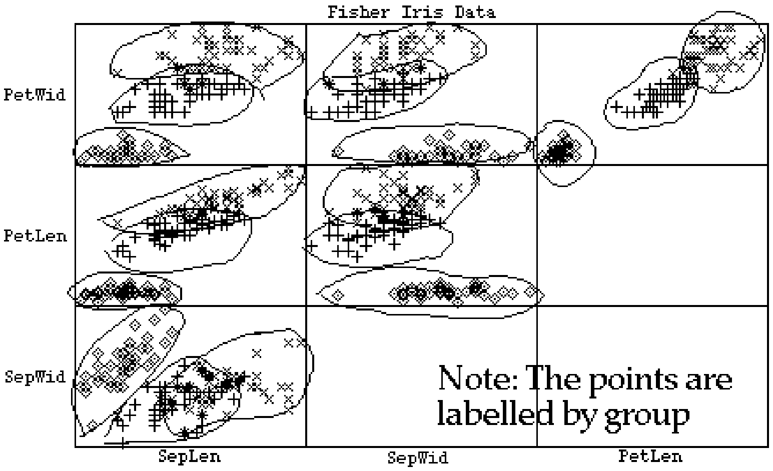
Note: The points are not labelled by group

There is clear bunching of points, but how many clusters are there?

Contrast with classification

In classification, you have known number g of "clusters", the *known* groups or populations. At both the training and validation stages, you know g and which group π_j each object belongs to.

Here are the same data, with each variety indicated by a unique symbol.



Note: The points are labelled by group

Is it possible to get a plot like this from data *without* variety information?

It may be, but you can never be sure it's right.

Mixture Model

When data consist of $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_N$, one possible model is that they are a <u>random sample</u> from a *mixture* of populations π_1, ..., π_g with

- mixture proportions (prior probabilities, prevalences)

$$p_1, \ldots, p_g$$

- distributions

$$f_1(\mathbf{x}),\ f_2(\mathbf{x}),\ \ldots,\ f_g(\mathbf{x}),$$

so that the distribution of $\mathbf{x}$ is

$$f(\mathbf{x}) = \sum_{1\le\ell\le g} p_\ell f_\ell(\mathbf{x})$$

Goals of cluster analysis might include

- Determine g = correct # of clusters
- Allocate each $\mathbf{x}_i$ to a cluster, all or most of whose members were sampled from the same π_j

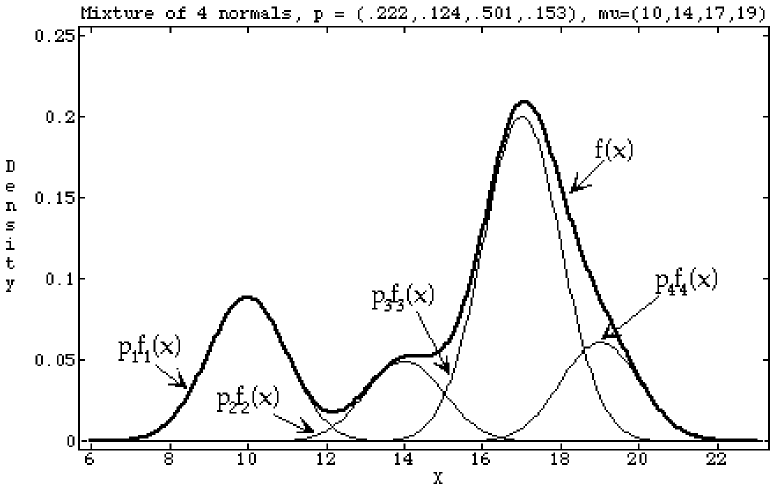
Possibly

- Estimate $f_\ell(\mathbf{x})$ and p_ℓ, $\ell = 1, \ldots, g$

Example: Suppose you know each $f_i(\mathbf{x})$ is $N_p(\boldsymbol{\mu}_i, \boldsymbol{\Sigma})$. In that case p_1,..., p_g, $\boldsymbol{\mu}_1$, ..., $\boldsymbol{\mu}_g$, and $\boldsymbol{\Sigma}$ would be unknown parameters to estimate from the sample.

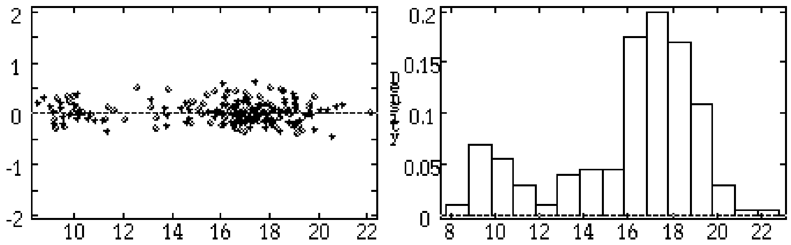
This may be very hard to do.

Particular case with p = 1, g = 4.



Even if you knew f(x) (heavy line) it would not be obvious this is a mixture of g = 4 populations.

Here are two representations of a sample of size N = 200 from this distribution.



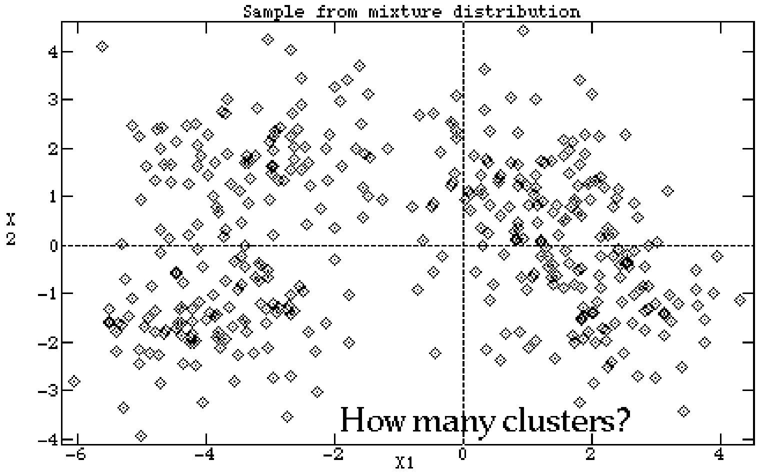
In the plot on the left, I "jittered" the data by adding random noise in the vertical direction. Only the left-right position is real data.

Jittering can be useful when there are many points that would otherwise overlap each other.

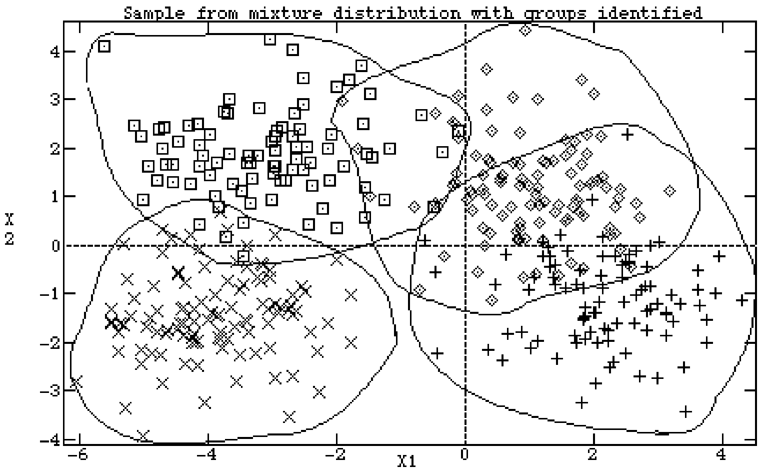
Even though there were g = 4 populations, it is very hard to see more that two clusters in either plot.

More usually, cluster analysis is exploratory, not based on an explicit model.

Plot of bivariate (p=2) data from mixture



How many clusters? 2? 3? 4? 1?



Actually a mixture of g = 4 populations.

Types of Data

- Data consists of $\mathbf{x}_1, \ldots, \mathbf{x}_N$, used to compute *dissimilarities* or *distances*

  $$d_{ij} = d(\mathbf{x}_i,\mathbf{x}_j),$$

  or *similarities*

  $$s_{ij} = s(\mathbf{x}_i,\mathbf{x}_j)$$

  where $d(\mathbf{x}_i,\mathbf{x}_j)$ or $s(\mathbf{x}_i,\mathbf{x}_j)$ is a specific function such as $d(\mathbf{x}_i,\mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|$.
- Data is an N by N matrix $\mathbf{D} = [d_{ij}]$ of **dissimilarity coefficients** between all pairs of N objects, without data pertaining to an individual object. In some cases $d_{ji} \neq d_{ij}$ and/or $d_{jj} \neq 0$.
- Data is an N by N matrix $\mathbf{S} = [s_{ij}]$ of **similarity coefficients** between all pairs of N objects. Higher s_{ij} means more similar. It can happen that $s_{ji} \neq s_{ij}$. Often $\left|s_{ij}\right| \leq 1$ with $s_{jj} = 1$.

When $\mathbf{S}$ or $\mathbf{D}$ is not symmetric ($s_{ij} \neq s_{ji}$ or $d_{ij} \neq d_{ji}$), one way to proceed is to symmetrize and use $\tilde{\mathbf{S}} = (\mathbf{S} + \mathbf{S}')/2$ ($\tilde{s}_{ij} = \tilde{s}_{ji} = (s_{ij}+s_{ji})/2$) or $\tilde{\mathbf{D}} = (\mathbf{D} + \mathbf{D}')/2$ ($\tilde{d}_{ij} = \tilde{d}_{ji} = (d_{ij}+d_{ji})/2$).

Example

A non expert tries to identify Morse encoded letters and numerals, 36 in all.

Suppose m_{ij} = # of identifications of symbol i as j, and n_i = number trials with symbol i used. Then

$$s_{ij} \equiv m_{ij}/n_i$$

is one measure of how easy it is to confuse the codes for symbols i and j, that is, how similar they are.

Here it can happen that $s_{ij} \neq s_{ji}$ and $s_{ii} \neq 1$.

Examples of distance measures

- **Euclidean**

$$d(\mathbf{x}_i,\mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{}\{(\mathbf{x}_i - \mathbf{x}_j)'(\mathbf{x}_i - \mathbf{x}_j)\}$$
$$= \sqrt{}\{\textstyle\sum_{1\le k\le p}(x_{ki} - x_{kj})^2\}$$

This is highly dependent scales of the x_k's.

- **Standardized Euclidean**

$$d(\mathbf{x}_i,\ \mathbf{x}_j) = \|\mathbf{x}_i^S - \mathbf{x}_j^S\|$$

where

$$\mathbf{x}_i^S = [z_{i1},\ \ldots,\ z_{ip}]',\ z_{ik} = (x_{ik} - \overline{x}_k)/\sqrt{s_{kk}}$$
$$\mathbf{x}_j^S = [z_{j1},\ \ldots,\ z_{jp}]',\ z_{jk} = (x_{jk} - \overline{x}_k)/\sqrt{s_{kk}}$$

are *standardized* versions of $\mathbf{x}_i$ and $\mathbf{x}_j$ using the same standard deviations $\sqrt{s_{kk}}$ for all cases.

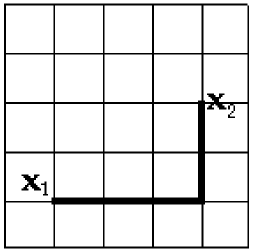
- **Generalized distance**

  $d(\mathbf{x}_i,\mathbf{x}_j) = \sqrt{}\{(\mathbf{x}_i-\mathbf{x}_j)'\mathbf{A}^{-1}(\mathbf{x}_i-\mathbf{x}_j)\}$, for some positive definite p×p $\mathbf{A}$.

  Some choices for $\mathbf{A}$ are $\mathbf{A} = \mathbf{S}$, where $\mathbf{S}$ is an overall variance matrix, or $\mathbf{A} = \mathbf{S}_{pooled}$ where $\mathbf{S}_{pooled}$ is pooled covariance matrix based on a preliminary clustering.

- **City block**

  $d(\mathbf{x}_i,\mathbf{x}_j) = \sum_{1\le k\le p}|x_{ki} - x_{kj}|$



$d(\mathbf{x}_i,\mathbf{x}_j) = 3 + 2 = 5$

$\|\mathbf{x}_i-\mathbf{x}_j\| = \sqrt{13} = 3.61$

- **Standardized city block** (better)

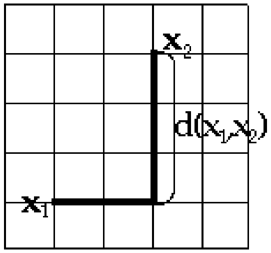
$$d(\mathbf{x}_i,\mathbf{x}_j) = \textstyle\sum_{1\le k\le p}|x_{ki} - x_{kj}|/\sqrt{s_{kk}}$$

- **Minkowsky distance**

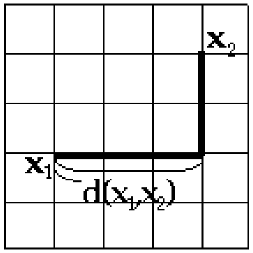
$$d(\mathbf{x}_i,\mathbf{x}_j) = \{\textstyle\sum_{1\le k\le p} \left|x_{ki} - x_{kj}\right|^m\}^{1/m}$$

Several others are special cases of Minkowsky distance:

m = 1 means city block.
m = 2 means Euclidean;
m = ∞ means $\max_k \left|x_{ki} - x_{kj}\right|$



$d(\mathbf{x}_i,\mathbf{x}_j) = 3$



$d(\mathbf{x}_i,\mathbf{x}_j) = 3$

- **Standardized Minkowsky**

$$d(\mathbf{x},\mathbf{y}) = \{\textstyle\sum_{1\le k\le p} \left|x_{ki} - x_{kj}\right|^m / s_{kk}^{m/2}\}^{1/m}$$

Clustering Variables

The data consist of N-dimensional vectors $\mathbf{X}_1, \ldots, \mathbf{X}_p$ to be divided into several sets, with each set consisting of "similar" variables.

Measures of similarity:

- $s(\mathbf{X},\mathbf{Y}) = r_{xy}$ (Pearson correlation)

$$r_{xy} = \sum(x-\overline{x})(y-\overline{y})/\{\sqrt{\sum(x-\overline{x})^2}\sqrt{\sum(y-\overline{y})^2}\}$$
$$= 1 - \|\mathbf{X}_s - \mathbf{Y}_s\|^2/(2(N-1))$$

where $\mathbf{X}_s$ and $\mathbf{Y}_s$ are standardized versions of $\mathbf{X}$ and $\mathbf{Y}$.

- $s(\mathbf{X},\mathbf{Y}) = \left|r_{xy}\right|$ is often more appropriate

r_{xy} is equivalent to the distance measure

$$d(\mathbf{X},\mathbf{Y}) \equiv \|\mathbf{X}_s - \mathbf{Y}_s\| = \sqrt{\{2(N-1)(1 - r_{xy})\}}$$

By analogy, $d(\mathbf{X},\mathbf{Y}) \equiv \sqrt{\{2(1 - \left|r_{xy}\right|)\}}$ would also be a natural distance measure.

For other purposes, you might evaluate dissimilarity in terms of how different their means and standard deviations were, say

$$d(\mathbf{X},\mathbf{Y}) \equiv \sqrt{\{(\overline{x} - \overline{y})^2 + (\log s_x/s_y)^2\}}$$
$$= \sqrt{\{(\overline{x} - \overline{y})^2 + (\log s_x - \log s_y)^2\}}.$$

You can always change a similarity coefficient into a dissimilarity coefficient, and vice versa, but not in a unique way:

Example: When an object is most similar to itself and has similarity 1 with itself, that is, $\max_\ell s_{k\ell} = s_{kk}$, both

$$d_{ij} = 1/s_{ij} - 1 = (1 - s_{ij})/s_{ij}$$

and

$$d_{ij} = \sqrt{\{2(1-s_{ij})\}}$$

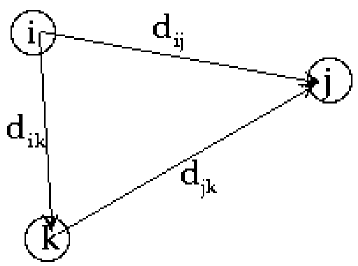
satisfy

$$d_{ii} = 0,\ d_{ij} \geq 0,\ i \neq j$$

and might be used as dissimilarities.

Dissimilarities d_{ij} may be "true distances" satisfying

- $d_{ij} = d_{ji}$ (symmetry)
- $d_{ii} = 0$
- $d_{ij} > 0,\ i \neq j$
- $d_{ij} \leq d_{ik} + d_{kj}$ (*triangle inequality*)



Triangle inequality means it's always shorter to go directly to j from i than to go via k.

These are valid for Minkowsky distance and $d_{ij} = \sqrt{\{(\mathbf{x}_i-\mathbf{x}_j)'\mathbf{A}^{-1}(\mathbf{x}_i-\mathbf{x}_j)\}}$. Hence they are valid for Euclidean and city block distances.

Note: You can do informative cluster analyses with dissimilarities that are not distances in this sense.

There are at least 3 general approaches to clustering.

1. **Agglomerative or combining**

- Start with a large number of "clusters", usually N, each consisting of a single object.
- Repeatedly merge "similar" or "neighboring" clusters, reducing the number of clusters after each merge.

2. **Divisive or dividing up**

- Start with a small number of clusters, often 1, consisting of all cases.
- Repeatedly split clusters into subclusters which are as dissimilar or distant as possible.

3. **Targeted number of clusters**:

- Specify in advance a number k of clusters and make an initial assignment of each case to a cluster.
- Repeatedly reassign objects from one cluster to another so that objects in each cluster become more similar and clusters become more different.

To start, you might partition case on the value of one variable or principal component, or just divide the objects arbitrarily in g equal parts. An example of this approach is *k-means clustering* (MacAnova function **kmeans()**).

4. **Estimation of mixture model**

- Postulate a parametric model for the different distributions, say MVN, possibly with some restriction on the $\mathbf{\Sigma}$'s and use parametric estimation. You might get starting values from an initial non-parametric clustering.