***Make Paired samples***

SPSS macros by Kirill Orlov

kior@akado.ru, ttnphns@gmail.com

<https://www.spsstools.net/en/KO-spssmacros>

All rights reserved

*Pairing cases of two samples.* Between two samples or sets optimal pairing is being done, such that the sum of within-pair differences gets minimized. Being used is Hungarian Algorithm for matching elements from two arrays into pairs.

# MACRO !KO\_HUNMATCH: OPTIMAL PAIRING VIA HUNGARIAN ALGORITHM

Version 2, Jun 2012 (Version 1, Apr 2010). Tested on SPSS 15, 17, 20.

!KO\_hunmatch prox= DISSIM /\*Values in dataset matrix are dissimilarities (DISSIM) or similarities (SIMIL)

/k= /\*How many pairs to create; if not specified then k = number of columns of input matrix

/licmat= 'd:\exercise\licmx.sav' /\*Optional: binary matrix sized as the input one,

/\*that prohibits to pair any rows with any columns – path/name of

/\*external sav-file, quoted/apostrophed

/rand= NO /\*Optionally: randomize choice in case of tied values:

/\*YES or NO (default).

Minimal specification PROX.

Hungarian algorithm also known as Kuhn-Munkres algorithm is probably the most well-known method for match-pairing objects from two sets which guarantees minimal overall dissimilarity between the paired objects. Consequently, this method is useful in survey and experimantal studies for the construction of a dataset consisting of pairs of maximally matched, similar individuals; a dataset to where one can apply statistical tests for paired (related) samples, such as paired Student’s t-test.

More generally, the algorithm is for optimal pairing of elements between two sets – whatever the sets represent. One set may be, for example, workers, and the other, say, jobs. And there is the matrix of values, each of which is the “cost” (whatever it might mean) of the assignment of the given job to the given worker. The task is to distribute jobs among workers – one man one job – so that the sum of the “costs” of the made assignments be absolutely minimal.

The algorithm gives this global optimal solution; however it is not very quick and is not for big data amounting to many thousands of objects in each of the two sets. Но он несравненно быстрее, чем попытка решить ту же задачу комбинаторным перебором.

The macro takes, as input, a rectangular matrix that is the working dataset, wherein rows are one sample of respondents (or one set of elements), and columns are the second sample (or set). Values in the matrix are dissimilarities or similarities between these and those (or the “cost” or “gain” of pairing these to those). Optionally you may specify a second matrix, identical by size to the first, and playing the role of “licensing” for the first one, in the sense – allows or prohibits to do these or those pairings. The macro returns a new unnamed dataset with the row numbers and the column numbers – the numbers of the objects which got paired by the algorithm.

Attention, if the proximity matrix is not square lay it so that there is *less columns than rows*. Similarly should be laid the optional licensing matrix. Missing values are not allowed anywhere.

Classic Hungarian algorithm pairs all the columns with some rows. The current macro is its more general modification allowing to request any number of pairs *k* (I’m grateful to A.G. McDowell, <http://www.mcdowella.demon.co.uk/>, who suggested the modification to me). It may seem paradoxical, but the less is *k* the greater the algorithm requaires time/memory resourses to work; but that is explained by the peculiarity of the modification. So, if you have very large input matrix and you need, on the contrary, only few pairs, the algorithm becomes inefficient (from the resourses standpoint). Then there might be better to use another matching algorithm, even simply selecting *k* minimal values in the matrix giving *k* completely different pairs; in the condition when *k* is small relative the size of the matrix such naive approach is reasonably probable to give the global, not a local, optimal pairing (which Hungarian algorithm gives always).

ПРИМЕР 1. Pairing of similar respondents of two subsamples. Qualitative and quantitative features participate. Qualitative features must completely match in two respondents in order to be allowed to pair them; quantitative features must differ as little as possible in order to pair two respondents.

proximities sex marriage /view=case /measure= euclid /print= none /matrix= out(\*).

select if caseno\_>50.

execute.

delete variables rowtype\_ to varname\_ var51 to var120.

recode all (0=1) (else=0).

save outfile= 'd:\exercise\licmat.sav'.

dataset activate data.

proximities item1 item2 item3 /view=case /measure= euclid /print= none /matrix= out(\*).

select if caseno\_>50.

execute.

delete variables rowtype\_ to varname\_ var51 to var120.

dataset name prox.

!KO\_hunmatch prox= DISSIM /licmat= 'd:\exercise\licmat.sav'.

* In the dataset (called *DATA*) there are 120 respondents. The first subsample is the first 50 cases, and the second is next 70 ones. Qualitative variables: *SEX, MARRIAGE* (gender – two categories, marital status – two categories); quantitative: *ITEM1, ITEM2, ITEM3* (responses to rating-scale items of a questionnaire).
* PROXIMITIES command outputs the matrix of euclidean distances between all the respondents by variables *SEX, MARRIAGE* in a new unnamed working dataset. SELECT IF removes the first 50 rows from it, and DELETE VARIABLES removes the last 70 columns of it, along with decoration variables from *ROWTYPE\_* to *VARNAME\_*. As the result, we have a matrix of 70 respondents of the 2nd subsample (rows) × 50 respondents of the 1st subsample (columns). Recode all exactly zero distances into 1, and rest into 0. We’ve got the “licensing matrix” which we save as an external file.
* Return to the initial data *DATA* (DATASET ACTIVATE). PROXIMITIES outputs the matrix of euclidean distances between all the respondents by variables *ITEM1, ITEM2, ITEM3* in a new unnamed working dataset. Next we turn this square matrix into the rectangular 70 × 50 one exactly like we did above. This matrix is the proximity matrix, the main one to enter the macro. Call it *PROX* (DATASET NAME).
* Running the macro, specifying that the values in the matrix are dissimilarities, and indicating the path to the external “licensing” matrix. The macro will do optimal pairing of all the 50 respondents-“columns” with the most similar to them 50 respondents out of the 70 respondents-“rows”.

***Subcommands***

**PROX**

Specify PROX=DISSIM if the input affinity matrix – the working rectangular dataset – contains dissimilarities (i.e. “cost of pairing”) between the rows and the columns. Or specify PROX=SIMIL if it contains similarities (i.e. “gain of pairing”) instead. Values in the matrix must be *positive*. If some values are not positive, add a positive number to make them all positive (the result of pairing does not depend on adding a constant).

**K**

How many “row-column” pairs with minimal overall within-pairs dissimilarity to create. Specify integer from 1 to the number of columns. By default K is taken as the number of columns.

**LICMAT**

Optional subcommand allowing to introduce a “licensing” matrix. This must be an external SAV file which is a dataset of the same size as the affinity matrix. In the licensing matrix only values 1 (pairing of the row and the column is allowed) and 0 (pairing of the row and the column is prohibited) are allowed.

**RAND**

This subcommand might be needed if there are many equal values (ties) in the affinity matrix, so that iy may become that more than two equally similar objects pretend to form a pair. With RAND=YES, in such situation the choice between such objects will be random (and may change from one to another run of the macro with the same input objects). While with RAND=NO the choice of pair from the pretendents is not random but depends on the order of rows and columns in the matrix.

***Special regimes***

The macro does not obey weighting and is not meant for split state of the dataset.