Package 'MConjoint'

February 19, 2015

Type Package

Title Conjoint Analysis through Averaging of Multiple Analyses

Version 0.1

Date 2013-05-14

Author William Hughes

Maintainer William Hughes <William.Hughes@rogers.com>

Description The package aids in creating a Conjoint Analysis design with extra cards. Unlike traditional ``holdout'' cards these cards are used to create a set of ``good'' (balanced and low correlation) designs. Each of these designs is analyzed and the average calculated.

License GPL-3

NeedsCompilation no

Repository CRAN

Date/Publication 2013-06-19 09:01:22

R topics documented:

`MConjoint-package` *Perform Conjoint Analysis using multiple designs*

Description

The Multiple Conjoint Analysis package changes the meaning and use of traditional holdout cases. Rather than using the holdout cases to check a single design, the "holdouts" are used to create a large set of designs, each of which is analyzed. The average result is used

Details

Package:	MConjoint
Type:	Package
Version:	0.1
Date:	2013-05-14
License:	GPL-3

The use of the routines centers around something I call a "despack" a design package. A despack contains despack$cards: a list of the m cards for which ranks are obtained; despack$designs: a list of designs each with n cards drawn from the list of m cards; despack$samples, a list of samples of length n, drawn from 1:m, corresponding to the cards used in the design; despack$coeffs: a list of matrices of linear coefficients; despack$all.utils: a list of lists of utility values, on for each column of the coeffs matrices; despack$all.imps, a list of matrices of importances, one column for each utility; despack$utils: a list of utilities (average taken over first index of the list of lists; despack$imps: the average of the list of importance matices

Start with a data set, full.design, with all possible cards. (This may be the full factorial design (all combinations of levels)) or some combinations may be removed.

Obtain a "good" design of n cards (for information on what makes a design good see the documentation for mc.good.desgins). To this you add extra.cards cards in such a way that you maximize the number of subsets of the m=n + extra.cards of length n that lead to "good" designs.

Both operations can be done by calling

```
orig.design = mc.get.initial.design(full.design)
```

`orig.design$design` will be the m cards for which you will collect data

You then obtain your data, `data`, a matrix with each column corresponding the the ranks given to the cards by one subject. Then run

```
despack = good.designs(orig.design$design)
```

This will give an initial despack, with $cards, $samples, and $designs

Fill the other elements of despack by calling

```
despack=M.Conjoint(despack,data)
```

This will print a summary with the utilites and the importances averaged over the subjects (an operation that may or may not be useful)

Author(s)

William Hughes

Maintainer: William Hughes <William.Hughes@rogers.com>

Examples

```
#  A simple conjoint problem.  Managers can make hiring descisions
#  based on the factors
#  University: Prestige, Excellent, Good; Sex: Male, Female;
#  Dress: smart, messy; Hair: long, short.
#  We want to determine the importance of these factors.
#  We interview two managers.  The first picks first by
#  University, then by sex, male over female, then
#  by dress, smart over messy, and does not care about hair
#  length.  The second is like the first except that
#  this manager picks female over male.

# start with the full factorial design

data(hire.candidates)

#get a questionaire

hire.questionaire = mc.get.initial.design(hire.candidates,max.trials=10)

#collect the data

data(hire.data)

#get a design pack for the analyis

hire.despack=mc.good.designs(hire.questionaire$design, size=20)

#do the conjoint analysis

hire.despack=M.Conjoint(hire.despack,hire.data)

# (note this illustrates the danger of averaging utilities.
# The average utility for both Male and Female is small, but
# Sex is important to both managers)
```

---

hire.candidates *All combinations of factors used for hiring*

Description

This is a full factorial (all combinations of factors) design.

Usage

```
data(hire.candidates)
```

Format

A data frame with 24 observations on the following 4 variables.

`University` a factor with levels `Prestige` `Excellent` `Good`

`Sex` a factor with levels `Male` `Female`

`Dress` a factor with levels `smart` `messy`

`Hair` a factor with levels `long` `short`

Details

We have a toy Conjoint Analysis problem for examples. We assume all hiring decisions are made on four factors. This is a data set with all 24 possible combinations.

Examples

```
data(hire.candidates)
```

---

hire.data *The Data for the Hiring Problem*

Description

A matrix with two columns each giving the ranks assigned by two mangers to the `hire.questionaire$design` cases. The managers choose first by university, then by Sex (the first chooses Male over Female, the second Female over Male), then by dress. Hair is ignored.

Usage

```
data(hire.data)
```

Format

A matrix of ranks with 12 rows and two columns

Examples

```
data(hire.data)
```

hire.despack *A Design Pack for the Hire Problem*

Description

This design pack contains 26 9 card designs and the corresponding line numbers from `hire.questionaires$design`.

Usage

```
data(hire.despack)
```

Format

The format is:
List of 2
$ samps :List of 26 vectors of 12 integres
$ designs:List of 26 data frames, each 9 obs. of 4 variables

Examples

```
data(hire.despack)
```

hire.questionaire *A Design for the Hiring Problem*

Description

This is a Conjoint Analysis design with 12 cards

Usage

```
data(hire.questionaire)
```

Format

The format is: List of 3
$ base.design:'data.frame': 9 obs. or 4 variables
$ added :'data.frame': 3 obs. of 4 variables
$ design :'data.frame': 12 obs. of 4 variables:

Examples

```
data(hire.questionaire)
```

M.Conjoint *M.Conjoint*

Description

Perform Conjoint Analysis by analyzing and averaging a number of designs

Usage

```
M.Conjoint(despack, data, type = "linear")
```

Arguments

despack	A despack with $cards, $samps and $designs filled
data	A matrix with each column the ranks given to despack$cards by one subject
type	a string indicating the method to use. Choose from linear

Details

This is a wrapper. It calls

```
mc.despack.linear.conjoint(despack,data)
mc.despack.linear.utils(despack)
mc.importances(despack)
mean.over.design.utils(despack$all.utils)
mean.over.design.imps(despack$all.imps)
```

To populate despack

It then calculates and outputs the average utilities and importances

Value

A fully populated design package (invisible)

Author(s)

William Hughes

Examples

```
data(hire.despack)
data(hire.data)
hire.despack=M.Conjoint(hire.despack,hire.data)
```

`mc.add.to.design` *Add cards to a design*

Description

Given a "good" design add some cards to get a new design

Usage

```
mc.add.to.design(all.possible, old.design, cards.to.add = 3, slack = 1,
                 tol = 0.2, max.trials = 100, max.good.designs=100)

mc.add.to.design.fast(all.possible, old.design, cards.to.add = 3, slack = 1, tol = 0.2)
```

Arguments

`all.possible`	all possible cards (can be all combinations of factors, or some can be removed)
`old.design`	An initial "good" design
`cards.to.add`	the number of cards to add
`slack`	How much the number of each factor can vary in a "good" design
`tol`	The largest cross correlation in a "good" design
`max.trials`	The maximum number of samples of extra.cards cards to look at.
`max.good.designs`	For any trial design, the maximum number of good designs to look for

Details

The function will take samples of size extra.cards from the set of cards all.possible-old.design and add them to old.design. For each sample the number of "good" (balanced an low cross correlation) designs of the same size as old.design is found. The sample which gives the largest number of "good" designs is used.

The default `max.trials` of 100 will lead to run times of several hours for largish problems. You can usually decrease this without losing too much. Increasing the number of trials may increase the number of good designs found, but usually not by much.

The default `max.good.designs` is set to 100. Analyzing more than this number of designs is unlikely to be needed outside of theoretical work.

mc.add.to.design.fast is useful for quick and dirty work

Value

`base.design`	The original old.design
`added`	The sample of cards added
`design`	The combined set of cards randomized

Author(s)

William Hughes

Examples

```
data(hire.candidates)
base.design = mc.get.one.design(hire.candidates, 9)

#use defaults, (except max.trials=10 for speed)

mc.add.to.design(hire.candidates,base.design, max.trials=10)

#add 4 cards, accepting cross corellations up to .35
#warning, this may take several minutes

#mc.add.to.design(hire.candidates,base.design,4,tol=.35)

# you can speed this up and in this case
# have almost as many good designs (53 vs.54)

#mc.add.to.design(hire.candidates,base.design,4,tol=.35,max.trials=10)
```

`mc.despack.linear.conjoint`

mc.despack.linear.conjoint

Description

Given a list of designs and the data find the coefficients of the linear fit for each design and each column of data.

Usage

```
mc.despack.linear.conjoint(despack, data = NULL)
```

Arguments

`despack`	a design pack with the fields $cards. $designs and possibly $samps
`data`	Each column consist of the ranks given to the m cards in despack$cards by a single subject.

Details

The function will only work correctly if the sample, despack$samps[i] correspond correctly to despack$desings[i] (eg. the sample consists of the corresponding row numbers from despack$cards). If despack$samps is null the the row names of despack$designs[i] are assumed to be the correct row numbers. (in practice this is often true)

Value

A despack is returned with the following fields filled

cards	copy of despack$cards
samps	copy of despack$samps
designs	copy of despack$designs
coeffs	the calculated coefficients

Author(s)

William Hughes

Examples

```
data(hire.despack)
data(hire.data)
hire.despack=mc.despack.linear.conjoint(hire.despack,hire.data)
hire.despack$coeffs[[3]]
```

`mc.despack.linear.utils` *mc.despack.linear.utils*

Description

Calculate the utilities from the linear fit coefficients

Usage

```
mc.despack.linear.utils(despack)
```

Arguments

despack	a despack with $coeffs filled

Value

a despack with all fields unchanged except that $all.utils will be replaced by the list of lists of calculated utilities

Author(s)

William Hughes

Examples

```
data(hire.despack)
data(hire.data)
hire.despack=mc.despack.linear.conjoint(hire.despack,hire.data)
hire.despack=mc.despack.linear.utils(hire.despack)
hire.despack$all.utils[[3]][[2]]
```

`mc.get.initial.design` *mc.get.initial.design*

Description

Given a set of cards, determine an initial design including extra cards.

Usage

```
mc.get.initial.design(full.design, cards = NULL, extra.cards = 3, slack = 1, tol = 0.2,
                      max.trials = 100)
```

Arguments

`full.design`	All cards (combinations of levels) that are deemed possible
`cards`	The number of cards in the base design. If this is null then the minimum possible number of cards + 3 (to allow for model fit issues) is used.
`extra.cards`	The number of extra cards added to the base design
`slack`	How much the number of each factor can vary in a "good" design
`tol`	The largest cross correlation in a "good" design
`max.trials`	The maximum number of samples of extra.cards cards to look at.

Details

This can be slow for big designs (lots of samples, a long time testing each one). You can set max.trials to limit the length of time the function runs. It will use the best sample it has seen.

For finer control call `mc.get.one.design` and `mc.add.to.design` separately.

Value

A design with cards + extra.cards rows

Author(s)

William Hughes

See Also

mc.get.one.design, mc.add.to.design

Examples

```
data(hire.candidates)

#default except max.trials=10 for speed

hire.questionaire = mc.get.initial.design(hire.candidates,max.trials=10)

#A base design of 10 cards with 5 extra cards and good cross correlations less than .17
#takes about 10 seconds

#hire.questionaire = mc.get.initial.design(hire.candidates,cards=10,extra.cards=5,tol=.17)
```

---

mc.get.one.design *mc.get.one.design*

Description

Get a single "good" design from a list of cards

Usage

```
mc.get.one.design(all.cards, cards, slack = 1, tol = 0.2, max.tries = 1e+06)
```

Arguments

all.cards	The set of cards from which the design is to be drawn
cards	The number of cards in the design.
slack	How much the number of each factor level can vary in a "good" design
tol	The largest cross correlation in a "good" design
max.tries	The maximum number of designs to look at.

Details

Take samples of size cards from all.cards to form designs. Check each of the designs to see if it is "good". Return the first good design found. (This is one very simple way of getting a design to start with. Alternatively you could use a classical design (e.g. a fractional factorial design) or an "optimal" design as produced by the **AlgDesign** package).

Value

A single design or NULL and an error message if no design is found

Author(s)

William Hughes

Examples

```
data(hire.candidates)

#get a nine card design
mc.get.one.design(hire.candidates,9)

#get a 15 card design with cross correlations less that .1
mc.get.one.design(hire.candidates,15,tol=.1)
```

---

mc.good.designs *mc.good.design*

Description

given a set of m cards, find "good" designs with cards rows

Usage

```
mc.good.designs(orig.set, cards = NULL, slack = 1, tol = 0.2, no.replace = TRUE,
                size = 100, max.trials = 1e+06)
```

Arguments

orig.set	a design of length m
cards	The number of cards in each "good" design found.
slack	How much the number of each factor can vary in a "good" design
tol	The largest cross correlation in a "good" design
no.replace	Sample without replacement: TRUE or FALSE
size	The number of "good" designs to find
max.trials	The maximum number of designs to look at

Details

The function takes samples with cards rows from the orig.design. For each sample it checks whether the design is "good". A design is said to be good if it is balanced (for each factor each level occurs about the same number of times, the maximum difference is slack) and the different factors are uncorrelated (maximum cross correlation is tol). Sampling continues (with or without replacement depending on no.replace) until one of size good designs are found, all designs have been checked, or max.trials designs have been checked. If fewer than size design are found then a warning is printed.

Value

A despack with the following field filled

cards	set equal to orig.set
samps	a list of samples, the row numbers of the corresponding designs
designs	the good designs found

Author(s)

William Hughes

Examples

```
data(hire.questionaire)


#default
mc.good.designs(hire.questionaire$design)

#look for 7 card designs, with the cross correlation tolerance increased to .3

#mc.good.designs(hire.questionaire$design,7,tol=.3)
```

---

mc.importances *mc.importances*

Description

Given a despack with $all.utils filled, calculate the importances

Usage

```
mc.importances(despack)
```

Arguments

despack	a design package with $all.utils filled

Details

Note that this function will only work if despack$utils is correctly filled. However, there is no requirement that the utilities come from a linear fit. The utilities are not checked against despack$coeffs which may not exist.

Value

a despack with all fields unchanged except that $all.imps will be replaced by the list of matrices of calculated utilities

Author(s)

William Hughes

Examples

```
data(hire.despack)
data(hire.data)
hire.despack=mc.despack.linear.conjoint(hire.despack,hire.data)
hire.despack=mc.despack.linear.utils(hire.despack)
hire.despack=mc.importances(hire.despack)
hire.despack$all.imps[12]
```

---

mc.mean.over.design.utils

mean over design functions

Description

Given the utilities or the importances calculate the average over all the designs

Usage

```
mc.mean.over.design.utils(all.utils)

mc.mean.over.design.imps(all.imps)
```

Arguments

all.utils	A list of lists of utilities, one list for each design
all.imps	A list of matrices of importances, one matrix for each design

Details

note the average importances are not the importances calculated from the average utilities

Value

a list of utilities or a matrix of importances

Author(s)

William Hughes

Examples

```
data(hire.despack)
data(hire.data)
hire.despack=mc.despack.linear.conjoint(hire.despack,hire.data)
hire.despack=mc.despack.linear.utils(hire.despack)
hire.despack=mc.importances(hire.despack)

mc.mean.over.design.utils(hire.despack$all.utils)
mc.mean.over.design.imps(hire.despack$all.imps)
```

Index