

Package ‘perturb’

January 24, 2012

Title Tools for evaluating collinearity

Version 2.04

Author John Hendrickx

Description ‘perturb’ evaluates collinearity by adding random noise to selected variables. ‘colldiag’ calculates condition numbers and variance decomposition proportions to test for collinearity and uncover its sources.

Maintainer John Hendrickx <John_Hendrickx@yahoo.com>

License GPL (>= 2)

Suggests car

URL <http://home.wanadoo.nl/john.hendrickx/statres/>

Repository CRAN

Date/Publication 2012-01-24 18:47:41

R topics documented:

colldiag	2
consumption	4
perturb	4
reclassify	8

Index	14
--------------	-----------

colldiag

*Condition indexes and variance decomposition proportions***Description**

Calculates condition indexes and variance decomposition proportions in order to test for collinearity among the independent variables of a regression model and identifies the sources of collinearity if present

Usage

```
colldiag(mod, scale = TRUE, center = FALSE, add.intercept = TRUE)

## S3 method for class 'colldiag'
print(x, dec.places=3, fuzz=NULL, fuzzchar=".", ...)
```

Arguments

mod	A model object or data-frame
scale	If FALSE, the data are left unscaled. Default is TRUE
center	If TRUE, data are centered. Default is FALSE
add.intercept	if TRUE, an intercept is added. Default is TRUE
x	A colldiag object
dec.places	number of decimal places to use when printing
fuzz	variance decomposition proportions less than <i>fuzz</i> are printed as <i>fuzzchar</i>
fuzzchar	character for small variance decomposition proportion values
...	arguments to be passed on to or from other methods

Details

Colldiag is an implementation of the regression collinearity diagnostic procedures found in Belsley, Kuh, and Welsch (1980). These procedures examine the “conditioning” of the matrix of independent variables.

Colldiag computes the condition indexes of the matrix. If the largest condition index (the condition number) is *large* (Belsley et al suggest 30 or higher), then there may be collinearity problems. All *large* condition indexes may be worth investigating.

Colldiag also provides further information that may help to identify the source of these problems, the *variance decomposition proportions* associated with each condition index. If a large condition index is associated two or more variables with *large* variance decomposition proportions, these variables may be causing collinearity problems. Belsley et al suggest that a *large* proportion is 50 percent or more.

Value

A colldiag object

condindx A vector of condition indexes

pi A matrix of variance decomposition proportions

print.colldiag prints the condition indexes as the first column of a table with the variance decomposition proportions beside them. print.colldiag has a fuzz option to suppress printing of small numbers. If fuzz is used, small values are replaced by a period ".". Fuzzchar can be used to specify an alternative character.

Note

Colldiag is based on the Stata program coldiag by Joseph Harkness <joe.harkness@jhu.edu>, Johns Hopkins University.

Author(s)

John Hendrickx <John_Hendrickx@yahoo.com>

References

D. Belsley, E. Kuh, and R. Welsch (1980). *Regression Diagnostics*. Wiley.

Belsley, D.A. (1991). *Conditioning diagnostics, collinearity and weak data in regression*. New York: John Wiley & Sons.

See Also

[lm](#), [scale](#), [svd](#), [\[car\]vif](#), [\[Design\]vif](#), [perturb](#)

Examples

```
# Belsley (1991). "Conditioning Diagnostics"
# The Consumption Function (pp. 149-154)
data(consumption)

ct1<-c(NA,c[-length(c)])

# compare (5.3)
m1<-lm(c~ct1+dpi+r+d_dpi)
anova(m1)
summary(m1)

# compare exhibit 5.11
cor(cbind(ct1,dpi,r,d_dpi),use="complete.obs")

# compare exhibit 5.12
cd<-colldiag(m1)
cd
print(cd,fuzz=.3)
```

consumption

The Consumption Function

Description

Example from pp 149-154 of Belsley (1991), Conditioning Diagnostics

Usage

```
data(consumption)
```

Format

A data frame with 28 observations on the following 5 variables.

year 1947 to 1974

c total consumption, 1958 dollars

r the interest rate (Moody's Aaa)

dpi disposable income, 1958 dollars

d\dpi annual change in disposable income

Source

pp 149-154 of Belsley (1991), Conditioning Diagnostics

References

Belsley, D.A. (1991). *Conditioning diagnostics, collinearity and weak data in regression*. New York: John Wiley & Sons.

Examples

```
data(consumption)
```

perturb

Perturbation analysis to evaluate collinearity

Description

Adds random noise to selected variables to evaluate collinearity. Also suitable for other models than linear regression and for categorical independent variables

Usage

```
perturb(mod, pvars = NULL, prange = NULL, ptrans = NULL,
        pfac = NULL, uniform = FALSE, niter = 100)
```

```
## S3 method for class 'perturb'
summary(object,dec.places=3,full=FALSE,...)
```

```
## S3 method for class 'perturb'
print.summary(x,...)
```

Arguments

mod	A model object, not necessarily type lm
pvars	Contains an array of variables to be perturbed. Random values are added to the variable, after which the model is re-estimated.
prange	Contains an array of values determining the magnitude of perturbations. There should be as many <i>prange</i> values as <i>pvars</i> variables.
ptrans	Contains an array of transformations to be applied. Each element should contain valid R syntax in quotes for deriving one of the <i>pvars</i> as a function of other variables.
pfac	Contains a list of categorical variables to be perturbed and parameters controlling the reclassification process. The first component must be a factor name. The name for the first component is ignored. Other list components should correspond with options for <code>reclassify</code> . The usage of these parameters is discussed more fully below under the heading “ <i>Categorical variables</i> ”.
uniform	If uniform=TRUE, random values from a uniform distribution $unif(-x/2, x/2)$ are added to the perturb variables, where x is the <i>prange</i> value corresponding with each <i>pvars</i> variable. The default is to add values from a normal distribution $N(0,x)$.
niter	Indicates the number of times to re-estimate the model. Default is 100.
object	a perturb object to be summarized
x	a summary.perturb object to be printed
dec.places	number of decimal places to use when printing
full	if TRUE, some extra information is printed
...	arguments to be passed on to or from other methods. Print options for class matrix may be used, e.g. print.gap

Details

Perturb is a tool for assessing the impact of small random changes (perturbations) to variables on parameter estimates. It is an alternative for collinearity diagnostics such as `vif` in the `car` package, `vif` in the `Design` package or `colldiag` in this package. Perturb is particularly useful for evaluating collinearity if interactions are present or nonlinear transformations of variables, e.g. a squared term. Perturb can show how the perturbations affect the estimates of a variable and terms derived from it whereas other collinearity diagnostics simply treat interactions and transformations as regular

independent variables. Perturb is not limited to linear regression but can be used for all regression-like models. Perturb can also deal with categorical variables by randomly misclassifying them to assess the impact on parameter estimates.

Perturb works by adding a small random “perturbation” value to selected independent variables, then re-estimating the model. This process is repeated *niter* times, after which a summary of the means, standard deviation, minimum and maximum of the parameter estimates is displayed. If collinearity is a serious problem in the data, then the estimates will be unstable and vary strongly.

Perturb can be used with categorical variables. Categorical variables are reclassified according to a table of reclassification probabilities. Random numbers determine to which category each case is reclassified at each iteration. The reclassification probabilities are specified to make reclassification to the same category highly likely. For ordered variables, short distance reclassification can be made more likely than long distance. See the section on “*Categorical variables*” and [reclassify](#) for further details.

Value

An object of class “perturb”. The main result is contained in `coef.table`, which contains the parameter estimates for each iteration of the perturbation analysis. `summary` prints the mean, standard deviation, minimum and maximum of `coef.table` over the iterations. If the option `full` is specified, [reclassify](#) prints additional information on how the reclassification probabilities were derived. Summary also prints information on the transformed model formula.

<code>coef.table</code>	Estimated coefficients for each iteration of the perturbation analysis
<code>formula</code>	The model formula used
<code>pvars</code>	The continuous variables perturbed in the analysis
<code>prange</code>	Magnitude of the perturbations
<code>ptrans2</code>	The transformations using the temporary variables of the perturbation analysis
<code>reclassify.tables</code>	objects produced by reclassify for each factor in the perturbation analysis
<code>formula2</code>	The model formula using temporary variables
<code>distribution</code>	“normal” or “uniform”, the distribution from which to draw random numbers

Categorical variables

In a perturbation analysis, categorical variables are reclassified with a high probability of remaining in the same category. This could be accomplished in several ways. [reclassify](#) lets you specify the target percentage reclassification, then adjusts the reclassification frequencies so that the expected frequencies of the reclassified variable are the same as those of the original. In addition, `reclassify` imposes a meaningful pattern of association between the original and the reclassified variable. See [reclassify](#) for details.

Categorical variables are specified in `perturb` as a list in the `pfac` option. The first (unnamed) component is the factor to be reclassified. The names of following components should be valid `reclassify` options followed by appropriate values. For example, to reclassify the factor “type” with a 95% probability of reclassify to the same category, use:

```
p2<-perturb(m2,pvars=c("income","education"),prange=c(1,1),
pfac=list("type",pcnt=95))
```

This command will iteratively re-estimate model `m2` 100 times (default). Random values from a normal distribution with a standard deviation of 1 will be added to the variables `income` and `education`. `Reclassify` creates a table of initial reclassification probabilities for `type` with a 95% probability of reclassification to the same category. This initial table is adjusted and the final reclassification probabilities printed in the output are subsequently used to reclassify `type` at each iteration.

Use a list of lists to specify a model with more than one reclassification factor, for example:

```
pfac=list(list("fegp6",pcnt=95),list("eyrc",pcnt=m1),list("expc",pcnt=m2))
q<-perturb(r1,pfac=pfac)
```

Note

`Perturb` can be used with estimation procedures other than `lm`. On the other hand, collinearity is a result of extreme (multiple) correlation among independent variables. Another option is `vif` in the **Design** package or `colldiag`, which only uses the independent variables of a regression model. This will usually be a faster solution since maximum likelihood procedures require iterative estimation for each iteration of the perturbation analysis. It is possible though that certain procedures are more sensitive to collinearity than `lm`, in which case `perturb` will be the preferred solution.

Author(s)

John Hendrickx <John_Hendrickx@yahoo.com>

References

- D. Belsley, E. Kuh, and R. Welsch (1980). *Regression Diagnostics*. Wiley.
- Belsley, D.A. (1991). *Conditioning diagnostics, collinearity and weak data in regression*. New York: John Wiley & Sons.
- Hendrickx, John, Ben Pelzer. (2004). *Collinearity involving ordered and unordered categorical variables*. Paper presented at the RC33 conference in Amsterdam, August 17-20 2004. Available at <http://www.xs4all.nl/~jhckx/perturb/>

See Also

`reclassify`, `colldiag`, `[car]vif`, `[Design]vif`

Examples

```
library(car)
data(Duncan)
attach(Duncan)
m1<-lm(prestige~income+education)
summary(m1)
anova(m1)
vif(m1)
p1<-perturb(m1,pvars=c("income","education"),prange=c(1,1))
summary(p1)
m2<-lm(prestige~income+education+type)
summary(m2)
anova(m2)
```

```

vif(m2)
p2<-perturb(m2,pvars=c("income","education"),prange=c(1,1),pfac=list("type",pcnt=95))
summary(p2)

## Not run:
r1<-lm(ses~fegp6+educyr+eyr+exp2)
summary(r1)
q<-perturb(r1,c("eyr","exp"),c(2.5,2.5),ptrans="exp2<-exp^2")
summary(q)

fegp6<-as.factor(fegp6)

# eyr and exp also as factors
eyrc<-cut(eyr,c(min(eyr),40,50,60,70,80,max(eyr)),include.lowest=T,right=F)
table(eyrc)
expc<-cut(exp,c(0,10,20,max(exp)),include.lowest=T,right=F)
table(expc)

# rough initial reclassification probabilities,
# program will ensure they sum to 100 row-wise
m1<-matrix(0,nlevels(eyrc),nlevels(expc))
m1[row(m1)==col(m1)]<-80
m1[abs(row(m1)-col(m1))==1]<-8
m1[abs(row(m1)-col(m1))==2]<-2
m1

m2<-matrix(0,nlevels(expc),nlevels(expc))
m2[row(m2)==col(m2)]<-80
m2[abs(row(m2)-col(m2))==1]<-10
m2[abs(row(m2)-col(m2))==2]<-2
m2

r2<-lm(ses~fegp6+eyrc+expc)
summary(r2)
pfac=list(list("fegp6",pcnt=95),list("eyrc",pcnt=m1),list("expc",pcnt=m2))
q2<-perturb(r2,pfac=pfac,niter=1)
summary(q2)

## End(Not run)

```

reclassify

Called by perturb to calculate reclassification tables

Description

reclassify is called by [perturb](#) to calculate reclassification probabilities for categorical variables. Use separately to experiment with reclassification probabilities.

Usage

```
reclassify(varname, pcnt = NULL, adjust = TRUE, bestmod = TRUE,
min.val = .1, diag = NULL, unif = NULL, dist = NULL, assoc = NULL)
```

```
## S3 method for class 'reclassify'
print(x, dec.places = 3, full = FALSE, ...)
```

Arguments

<code>varname</code>	a factor to be reclassified
<code>pcnt</code>	initial reclassification percentages
<code>adjust</code>	makes the expected frequency distribution of the reclassified variable equal to that of the original
<code>bestmod</code>	imposes an appropriate pattern of association between the original and the reclassified variable
<code>min.val</code>	value to add to empty cells of the initial expected table when estimating the best model
<code>diag</code>	The odds of same versus different category reclassification
<code>unif</code>	Controls short distance versus long distance reclassification for ordered variables
<code>dist</code>	alternative parameter for short versus long distance reclassification
<code>assoc</code>	a matrix defining a loglinear pattern of association
<code>x</code>	a reclassify object to be printed
<code>dec.places</code>	number of decimal places to use when printing
<code>full</code>	if TRUE, some extra information is printed
<code>...</code>	arguments to be passed on to or from other methods. Print options for class <code>matrix</code> may be used, e.g. <code>print.gap</code>

Details

`reclassify` creates a table of reclassification probabilities for *varname*. By default, the reclassification probabilities are defined so that the expected frequency distribution of the reclassified variable is identical to that of the original. In addition, a meaningful pattern of association is imposed between the original and the reclassified variable. `reclassify` is called by `perturb` to calculate reclassification probabilities for categorical variables. `reclassify` can be used separately to find a suitable reclassification probabilities.

Reclassify has several options but the most relevant will generally be the `pcnt` option. The argument for `pcnt` can be

- a scalar
- a vector of length n
- a vector of length n^2 , where n is the number of categories of the variable to be reclassified.

If the argument for `pcnt` is a scalar, its value is taken to be the percentage of cases to be reclassified to the same category, which is the same for all categories. A table of initial reclassification probabilities for the original by the reclassified variable is created with this value divided by 100 on the diagonal and equal values on off-diagonal cells.

If the argument for `pcnt` is a vector of length n , its values indicate the percentage to be reclassified to the same category for each category separately. These values divided by 100 form the diagonal of the table of initial reclassification probabilities. Off-diagonal cells have the same values for rows so that the row sum is equal to 1.

If the argument for `pcnt` is a vector of length n^2 , its values form the table of initial reclassification probabilities. `prop.table` is used to ensure that these values sum to 1 over the columns. Specifying a complete table of initial reclassification probabilities will be primarily useful when an ordered variable is being reclassified.

`Reclassify` prints an initial table of reclassification probabilities based on the `pcnt` option. This table is not used directly though but *adjusted* to make the expected frequencies of the reclassified variable identical to those of the original. In addition, a meaningful pattern of association is imposed between the original and the reclassified variable. Details are given in the section “*Adjusting the reclassification probabilities*”.

Knowledgeable users can specify a suitable pattern of association directly, bypassing the `pcnt` option. Details are given in the section “*Specifying a pattern of association directly*”.

Value

An object of class `reclassify`. By default, `print.reclassify` prints the variable name and the `reclass.prob`. If the full option is used with `print.reclassify`, additional information such as the initial reclassification probabilities, initial expected table, best model, are printed as well.

<code>variable</code>	The variable specified
<code>reclass.prob</code>	Row-wise proportions of <code>fitted.table</code>
<code>cum.reclass.prob</code>	Cumulative row-wise proportions
<code>exptab\$init.pcnt</code>	initial reclassification probabilities (option <code>pcnt</code>)
<code>exptab\$init.tbl</code>	initial expected frequencies (option <code>pcnt</code>)
<code>bestmod</code>	The best model found for the table of initial expected frequencies (option <code>pcnt</code>)
<code>assoc</code>	The log pattern of association specified using <code>pcnt</code> and <code>bestmod=FALSE</code>
<code>coef</code>	The coefficients of a fitted loglinear model
<code>fitted.table</code>	The adjusted table of expected frequencies

Adjusting the reclassification probabilities

A problem with the initial reclassification probabilities created using `pcnt` is that the expected frequencies of the reclassified variable will not be the same as those of the original. Smaller categories will become larger in the expected frequencies, larger categories will become smaller. This can

be seen in the column marginal of the initial table of expected frequencies in the `reclassify` output. This could have a strong impact on the standard errors of reclassified variables, particularly as categories differ strongly in size.

To avoid this, the initial expected table is *adjusted* so that the column margin is the same as the row margin, i.e. the expected frequencies of the reclassified variable are the same as those of the original. Use `adjust=FALSE` to skip this step. In that case the initial reclassification probabilities are also the final reclassification probabilities.

A second objection to the initial reclassification probabilities is that the pattern of association between the original and the reclassified variable is arbitrary. The association between some combinations of categories is higher than for others. `Reclassify` therefore derives an appropriate pattern of association for the initial expected table of the original by reclassified variable. This pattern of association is used when “adjusting” the marginals to make the frequency distribution of the reclassified variable identical to that of the original. Use the option `bestmod=FALSE` to skip this step.

The patterns of association used by `reclassify` are drawn from loglinear models for square tables, also known as “mobility models” (Goodman 1984, Hout 1983). Many texts on loglinear modelling contain a brief discussion of such models as well. For unordered variables, a “quasi-independent” pattern of association would be appropriate. Under quasi-independent association, the row variable is independent of the column variable if the diagonal cells are ignored.

If the argument for `pcnt` was a scalar, `reclassify` fits a “quasi-independent (constrained)” model. This model has a single parameter `diag` which indicates the log-odds of same versus different reclassification. This log-odds is the same for all categories. If the argument was of vector of length n , then a regular quasi-independence model is fitted with parameters `diag1` to `diagn`. These parameters indicate the log-odds of same versus different category reclassification, which is different for each category. For both models, the reclassified category is independent of the original category if the diagonal cells are ignored.

If the argument for `pcnt` was a vector of length n^2 , `reclassify` fits two models, a “quasi-distance model” and a “quasi-uniform association” model, and selects the one with the best fit to the initial expected table. Both have the `diag` parameter of the “quasi-independence (constrained)” model. An additional parameter is added to make short distance reclassification more likely than long distance reclassification. The quasi-uniform model is stricter: it makes reclassification less likely proportionately to the squared difference between the two categories. The distance model makes reclassification less likely proportionately to the absolute difference between the two categories.

In some cases, the initial expected table based on the `pcnt` option contains empty cells. To avoid problems when estimating the best model for this table, a value of `.1` is added to these cells. Use the `min.val` option to specify a different value.

Specifying a pattern of association directly

If the `pcnt` option is used, `reclassify` automatically determines a suitable pattern of association between the original and the reclassified variable. Knowledgeable users can also specify a pattern of association directly. The final reclassification probabilities will then be based on these values. Built-in options for specifying the loglinear parameters of selected mobility models are:

diag quasi-independence constrained (same versus different category reclassification)

unif uniform association (long versus short distance reclassification for ordered categories)

dist linear distance model (allows more long distance reclassification than uniform association)

The `assoc` option can be used to specify an association pattern of one's own choice. The elements of `assoc` should refer to matrices with an appropriate loglinear pattern of association. Such matrices can be created in many ways. An efficient method is:

```
wrk<-diag(table(factor))
myassoc<-abs(row(wrk)-col(wrk))*-log(5)
```

This creates a square diagonal matrix called `wrk` with the same number of rows and columns as the levels of `factor`. `row(wrk)` and `col(wrk)` can now be used to define a loglinear pattern of association, in this case a distance model with parameter 5. `reclassify` checks the length of the matrix equals n^2 , where n is the number of categories of `varname` and ensures that the pattern of association is symmetric.

Imposing a pattern of association

A table with given margins and a given pattern of association can be created by

- estimating a loglinear model of independence for a table with the desired margins
- while specifying the log pattern of association as an offset variable (cf. Kaufman & Schervish (1986), Hendrickx (2004)).

The body of the table is unimportant as long as it has the appropriate margins. The predicted values of the model form a table with the desired properties.

The expected table of the original by the reclassified variable is adjusted by creating a table with the frequency distribution of the original variable on the diagonal cells. This table then has the same marginals for the row and column variables. The pattern of association is determined by the `reclassify` options. If `pcnt` is used and `bestmod=TRUE` then the predicted values of the best model are used as the offset variable. If `bestmod=FALSE`, the log values of the initial expected table are made symmetric and used as the offset variable. If a loglinear model was specified directly, a variable is created in the manner of the `assoc` example.

A small modification in procedure is that `reclassify` uses a model of equal main effects rather than independence. Since the pattern of association is always symmetric, the created table will then also be exactly symmetric with the frequency distribution of the original variable as row and column marginal.

Changes from version 1

Version 1 was not made available from CRAN and so I felt justified in not making version 2 entirely backward compatible. The `misclass` option has been dropped; replace `misclass=5` by `pcnt=95,adjust=FALSE`. Replace `q=2` by `diag=log(2)` and `u=3` by `unif=log(3)`.

Author(s)

John Hendrickx <John_Hendrickx@yahoo.com>

References

Goodman, Leo A. (1984). *The analysis of cross-classified data having ordered categories*. Cambridge, Mass.: Harvard University Press.

Hendrickx, J. (2004). Using standardised tables for interpreting loglinear models. *Quality & Quantity* 38: 603-620.

Hendrickx, John, Ben Pelzer. (2004). *Collinearity involving ordered and unordered categorical variables*. Paper presented at the RC33 conference in Amsterdam, August 17-20 2004. Available at <http://www.xs4all.nl/~jhckx/perturb/>

Hout, M. (1983). *Mobility tables*. Beverly Hills: Sage Publications.

Kaufman, R.L., & Schervish, P.G. (1986). Using adjusted crosstabulations to interpret log-linear relationships. *American Sociological Review* 51:717-733

See Also

[perturb](#), [colldiag](#), [\[car\]vif](#), [\[Design\]vif](#)

Examples

```
library(car)
data(Duncan)
attach(Duncan)

reclassify(type,pcnt=95)
```

Index

*Topic **datasets**

consumption, 4

*Topic **regression**

colldiag, 2

perturb, 4

reclassify, 8

colldiag, 2, 5, 7, 13

consumption, 4

lm, 3, 7

perturb, 3, 4, 8, 9, 13

print.colldiag (colldiag), 2

print.reclassify (reclassify), 8

print.summary.perturb (perturb), 4

prop.table, 10

reclassify, 5–7, 8

scale, 3

summary.perturb (perturb), 4

svd, 3

vif, 3, 5, 7, 13