Package ‘rms’

May 9, 2026

Version 8.1-1

Date 2026-02-17

Title Regression Modeling Strategies

Depends R (>= 4.4.0), Hmisc (>= 5.2-5)

Imports methods, survival, quantreg, ggplot2, Matrix, SparseM, rpart, nlme (>= 3.1-123), polspline, multcomp, htmlTable (>= 1.11.0), htmltools, MASS, cluster, digest, colorspace, knitr, grDevices, scales

Suggests boot, tcltk, plotly (>= 4.5.6), mice, icenReg, rmsb, nnet, VGAM, lattice, kableExtra

Description Regression modeling, testing, estimation, validation, graphics, prediction, and typesetting by storing enhanced model design attributes in the fit. 'rms' is a collection of functions that assist with and streamline modeling. It also contains functions for binary and ordinal logistic regression models, ordinal models for continuous Y with a variety of distribution families, and the Buckley-James multiple regression model for right-censored responses, and implements penalized maximum likelihood estimation for logistic and ordinary linear models. 'rms' works with almost any regression model, but it was especially written to work with binary or ordinal regression models, Cox regression, accelerated failure time models, ordinary linear models,the Buckley-James model, generalized least squares for serially or spatially correlated observations, generalized linear models, and quantile regression.

License GPL (>= 2)

URL https://hbiostat.org/R/rms/, https://github.com/harrelfe/rms

LazyLoad yes

RoxygenNote 7.3.3

NeedsCompilation yes

Author Frank E Harrell Jr [aut, cre]

Maintainer Frank E Harrell Jr <fh@fharrell.com>

Repository CRAN

Date/Publication 2026-02-18 06:30:02 UTC

Contents

adapt_orm . . . 4
anova.rms . . . 5
as.data.frame.Ocens . . . 12
bj . . . 13
bootBCa . . . 17
bootcov . . . 18
bplot . . . 27
calibrate . . . 30
contrast.rms . . . 34
cph . . . 43
cr.setup . . . 50
datadist . . . 52
ExProb . . . 55
fastbw . . . 57
Function . . . 59
gendata . . . 61
ggplot.npsurv . . . 63
ggplot.Predict . . . 65
gIndex . . . 73
Glm . . . 77
Gls . . . 79
groupkm . . . 81
hazard.ratio.plot . . . 83
ie.setup . . . 85
impactPO . . . 86
importedexported . . . 89
infoMxop . . . 90
intCalibration . . . 92
is.na.Ocens . . . 94
latex.cph . . . 94
latexrms . . . 96
lrm . . . 98
lrm.fit . . . 107
LRupdate . . . 111
matinv . . . 112
nomogram . . . 114
npsurv . . . 121
Ocens . . . 123
Ocens2ord . . . 123
Ocens2Surv . . . 126
Olinks . . . 126
ols . . . 127
ordESS . . . 130
ordParallel . . . 131
orm . . . 133
orm.fit . . . 141

pentrace . . . 144
plot.contrast.rms . . . 148
plot.Predict . . . 149
plot.rexVar . . . 157
plot.xmean.ordinaly . . . 159
plotIntercepts . . . 161
plotp.Predict . . . 162
poma . . . 165
pphsm . . . 166
predab.resample . . . 167
Predict . . . 171
predict.lrm . . . 177
predictrms . . . 180
print.cph . . . 187
print.Glm . . . 188
print.impactPO . . . 189
print.Ocens . . . 189
print.ols . . . 190
print.rexVar . . . 191
prmiInfo . . . 192
processMI . . . 193
processMI.fit.mult.impute . . . 193
psm . . . 195
Punits . . . 200
recode2integer . . . 201
residuals.cph . . . 202
residuals.Glm . . . 204
residuals.lrm . . . 205
residuals.ols . . . 210
rexVar . . . 212
rms . . . 213
rms.trans . . . 216
rmsMisc . . . 219
rmsOverview . . . 226
robcov . . . 236
Rq . . . 239
sensuc . . . 241
setPb . . . 245
specs.rms . . . 247
summary.rms . . . 248
survest.cph . . . 252
survest.orm . . . 255
survest.psm . . . 256
survfit.cph . . . 258
survplot . . . 259
survplot.orm . . . 266
val.prob . . . 269
val.surv . . . 273

validate 278
validate.cph 280
validate.lrm 283
validate.ols 286
validate.rpart 287
validate.Rq 289
vif 291
which.influence 292
Xcontrast 293
[.Ocens 294

Index **296**

`adapt_orm` *Adaptive orm Fit For a Single Continuous Predictor*

Description

Finds the best fitting `orm` model for a single continuous predictor `x`, starting by finding the link function with the smallest deviance when the predictor is modeled with 4 knots in a restricted cubic spline function. Then the function finds the number of knots minimizing AIC when this link function is used. Candidate number of knots are 0 (linear fit), 3, 4, 5, `maxk`.

Usage

```
adapt_orm(x, y, maxk = 6, ...)
```

Arguments

`x`	a numeric vector
`y`	a numeric or factor variable representing an ordinal dependent variable, or an `Ocens` object
`maxk`	maximum number of knots to try
`...`	arguments to orm other than `family`, `x`, `y`

Value

the best `orm` fit object

Author(s)

Frank Harrell

Examples

```
## Not run:
f <- adapt_orm(age, blood_pressure)
f$stats['d.f.']   # print no. of parameters for age
f$family          # print optimum link found

## End(Not run)
```

---

anova.rms *Analysis of Variance (Wald, LR, and F Statistics)*

Description

The anova function automatically tests most meaningful hypotheses in a design. For example, suppose that age and cholesterol are predictors, and that a general interaction is modeled using a restricted spline surface. anova prints Wald statistics (F statistics for an ols fit) for testing linearity of age, linearity of cholesterol, age effect (age + age by cholesterol interaction), cholesterol effect (cholesterol + age by cholesterol interaction), linearity of the age by cholesterol interaction (i.e., adequacy of the simple age * cholesterol 1 d.f. product), linearity of the interaction in age alone, and linearity of the interaction in cholesterol alone. Joint tests of all interaction terms in the model and all nonlinear terms in the model are also performed. For any multiple d.f. effects for continuous variables that were not modeled through rcs, pol, lsp, etc., tests of linearity will be omitted. This applies to matrix predictors produced by e.g. poly or ns.

For lrm, orm, cph, psm and Glm fits, the better likelihood ratio chi-square tests may be obtained by specifying test='LR'. Fits must use x=TRUE, y=TRUE to run LR tests. The tests are run fairly efficiently by subsetting the design matrix rather than recreating it.

print.anova.rms is the printing method. plot.anova.rms draws dot charts depicting the importance of variables in the model, as measured by Wald or LR χ^2, χ^2 minus d.f., AIC, P-values, partial R^2, R^2 for the whole model after deleting the effects in question, or proportion of overall model R^2 that is due to each predictor. latex.anova.rms is the latex method. It substitutes Greek/math symbols in column headings, uses boldface for TOTAL lines, and constructs a caption. Then it passes the result to latex.default for conversion to LaTeX.

When the anova table was converted to account for missing data imputation by processMI, a separate function prmiInfo can be used to print information related to imputation adjustments.

For Bayesian models such as blrm, anova computes relative explained variation indexes (REV) based on approximate Wald statistics. This uses the variance-covariance matrix of all of the posterior draws, and the individual draws of betas, plus an overall summary from the posterior mode/mean/median beta. Wald chi-squares assuming multivariate normality of betas are computed just as with frequentist models, and for each draw (or for the summary) the ratio of the partial Wald chi-square to the total Wald statistic for the model is computed as REV.

The print method calls latex or html methods depending on options(prType=). For latex a table environment is not used and an ordinary tabular is produced. When using html with Quarto or RMarkdown, results='asis' need not be written in the chunk header.

html.anova.rms just calls latex.anova.rms.

Usage

```
## S3 method for class 'rms'
anova(object, ..., main.effect=FALSE, tol=.Machine$double.eps,
      test=c('F','Chisq','LR'), india=TRUE, indnl=TRUE, ss=TRUE,
      vnames=c('names','labels'),
      posterior.summary=c('mean', 'median', 'mode'), ns=500, cint=0.95,
      fitargs=NULL)

## S3 method for class 'anova.rms'
print(x,
      which=c('none','subscripts','names','dots'),
      table.env=FALSE, ...)

## S3 method for class 'anova.rms'
plot(x,
     what=c("chisqminusdf","chisq","aic","P","partial R2","remaining R2",
            "proportion R2", "proportion chisq"),
     xlab=NULL, pch=16,
     rm.totals=TRUE, rm.ia=FALSE, rm.other=NULL, newnames,
     sort=c("descending","ascending","none"), margin=c('chisq','P'),
     pl=TRUE, trans=NULL, ntrans=40, height=NULL, width=NULL, ...)

## S3 method for class 'anova.rms'
latex(object, title, dec.chisq=2,
      dec.F=2, dec.ss=NA, dec.ms=NA, dec.P=4, dec.REV=3,
      table.env=TRUE,
      caption=NULL, fontsize=1, params, ...)

## S3 method for class 'anova.rms'
html(object, ...)
```

Arguments

`object`	a `rms` fit object. `object` must allow `vcov` to return the variance-covariance matrix. For `latex` is the result of `anova`.
`...`	If omitted, all variables are tested, yielding tests for individual factors and for pooled effects. Specify a subset of the variables to obtain tests for only those factors, with a pooled tests for the combined effects of all factors listed. Names may be abbreviated. For example, specify `anova(fit,age,cholesterol)` to get a Wald statistic for testing the joint importance of age, cholesterol, and any factor interacting with them. Add `test='LR'` to get a likelihood ratio chi-square test instead. Can be optional graphical parameters to send to `dotchart2`, or other parameters to send to `latex.default`. Ignored for `print`. For `html.anova.rms` the arguments are passed to `latex.anova.rms`.
`main.effect`	Set to `TRUE` to print the (usually meaningless) main effect tests even when the factor is involved in an interaction. The default is `FALSE`, to print only the effect of the main effect combined with all interactions involving that factor.

`tol` singularity criterion for use in matrix inversion

`test` For an `ols` fit, set `test="Chisq"` to use Wald χ^2 tests rather than F-tests. For `lrm, orm, cph, psm` and `Glm` fits set `test='LR'` to get likelihood ratio χ^2 tests. This requires specifying `x=TRUE, y=TRUE` when fitting the model.

`india` set to `FALSE` to exclude individual tests of interaction from the table

`indnl` set to `FALSE` to exclude individual tests of nonlinearity from the table

`ss` For an `ols` fit, set `ss=FALSE` to suppress printing partial sums of squares, mean squares, and the Error SS and MS.

`vnames` set to `'labels'` to use variable labels rather than variable names in the output

`posterior.summary` specifies whether the posterior mode/mean/median beta are to be used as a measure of central tendence of the posterior distribution, for use in relative explained variation from Bayesian models

`ns` number of random samples from the posterior draws to use for REV highest posterior density intervals

`cint` HPD interval probability

`fitargs` a list of extra arguments to be passed to the fitter for LR tests

`x` for `print,plot,text` is the result of `anova`.

`which` If `which` is not `"none"` (the default), `print.anova.rms` will add to the rightmost column of the output the list of parameters being tested by the hypothesis being tested in the current row. Specifying `which="subscripts"` causes the subscripts of the regression coefficients being tested to be printed (with a subscript of one for the first non-intercept term). `which="names"` prints the names of the terms being tested, and `which="dots"` prints dots for terms being tested and blanks for those just being adjusted for.

`what` what type of statistic to plot. The default is the χ^2 statistic for each factor (adding in the effect of higher-ordered factors containing that factor) minus its degrees of freedom. The R2 choices for `what` only apply to `ols` models.

`xlab` x-axis label, default is constructed according to `what`. `plotmath` symbols are used for R, by default.

`pch` character for plotting dots in dot charts. Default is 16 (solid dot).

`rm.totals` set to `FALSE` to keep total χ^2s (overall, nonlinear, interaction totals) in the chart.

`rm.ia` set to `TRUE` to omit any effect that has `"*"` in its name

`rm.other` a list of other predictor names to omit from the chart

`newnames` a list of substitute predictor names to use, after omitting any.

`sort` default is to sort bars in descending order of the summary statistic. Available options: 'ascending', 'descending', 'none'.

`margin` set to a vector of character strings to write text for selected statistics in the right margin of the dot chart. The character strings can be any combination of `"chisq"`, `"d.f."`, `"P"`, `"partial R2"`, `"proportion R2"`, and `"proportion chisq"`. Default is to not draw any statistics in the margin. When `plotly` is in effect, margin values are instead displayed as hover text.

pl	set to FALSE to suppress plotting. This is useful when you only wish to analyze the vector of statistics returned.
trans	set to a function to apply that transformation to the statistics being plotted, and to truncate negative values at zero. A good choice is trans=sqrt.
ntrans	n argument to pretty, specifying the number of values for which to place tick marks. This should be larger than usual because of nonlinear scaling, to provide a sufficient number of tick marks on the left (stretched) part of the chi-square scale.
height, width	height and width of plotly plots drawn using dotchartp, in pixels. Ignored for ordinary plots. Defaults to minimum of 400 and 100 + 25 times the number of test statistics displayed.
title	title to pass to latex, default is name of fit object passed to anova prefixed with "anova.". For Windows, the default is "ano" followed by the first 5 letters of the name of the fit object.
dec.chisq	number of places to the right of the decimal place for typesetting χ^2 values (default is 2). Use zero for integer, NA for floating point.
dec.F	digits to the right for F statistics (default is 2)
dec.ss	digits to the right for sums of squares (default is NA, indicating floating point)
dec.ms	digits to the right for mean squares (default is NA)
dec.P	digits to the right for P-values
dec.REV	digits to the right for REV
table.env	see latex
caption	caption for table if table.env is TRUE. Default is constructed from the response variable.
fontsize	font size for html output; default is 1 for 1em
params	used internally when called through print.

Details

If the statistics being plotted with plot.anova.rms are few in number and one of them is negative or zero, plot.anova.rms will quit because of an error in dotchart2.

The latex method requires LaTeX packages relsize and needspace.

Value

anova.rms returns a matrix of class anova.rms containing factors as rows and χ^2, d.f., and P-values as columns (or d.f., partial SS, MS, F, P). An attribute vinfo provides list of variables involved in each row and the type of test done. plot.anova.rms invisibly returns the vector of quantities plotted. This vector has a names attribute describing the terms for which the statistics in the vector are calculated.

Side Effects

print prints, latex creates a file with a name of the form "title.tex" (see the title argument above).

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

prmiInfo, rms, rmsMisc, lrtest, rms.trans, summary.rms, plot.Predict, ggplot.Predict, solvet, locator, dotchart2, latex, xYplot, anova.lm, contrast.rms, pantext

Examples

```
require(ggplot2)
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
treat <- factor(sample(c('a','b','c'), n,TRUE))
num.diseases <- sample(0:4, n,TRUE)
age <- rnorm(n, 50, 10)
cholesterol <- rnorm(n, 200, 25)
weight <- rnorm(n, 150, 20)
sex <- factor(sample(c('female','male'), n,TRUE))
label(age) <- 'Age'      # label is in Hmisc
label(num.diseases) <- 'Number of Comorbid Diseases'
label(cholesterol) <- 'Total Cholesterol'
label(weight) <- 'Weight, lbs.'
label(sex) <- 'Sex'
units(cholesterol) <- 'mg/dl'   # uses units.default in Hmisc


# Specify population model for log odds that Y=1
L <- .1*(num.diseases-2) + .045*(age-50) +
     (log(cholesterol - 10)-5.2)*(-2*(treat=='a') +
     3.5*(treat=='b')+2*(treat=='c'))
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)


fit <- lrm(y ~ treat + scored(num.diseases) + rcs(age) +
               log(cholesterol+10) + treat:log(cholesterol+10),
           x=TRUE, y=TRUE)   # x, y needed for test='LR'
a <- anova(fit)                       # Test all factors
b <- anova(fit, treat, cholesterol)   # Test these 2 by themselves
                                      # to get their pooled effects
a
b
a2 <- anova(fit, test='LR')
b2 <- anova(fit, treat, cholesterol, test='LR')
a2
b2

# Add a new line to the plot with combined effects
s <- rbind(a2, 'treat+cholesterol'=b2['TOTAL',])
```

```
class(s) <- 'anova.rms'
plot(s, margin=c('chisq', 'proportion chisq'))

g <- lrm(y ~ treat*rcs(age))
dd <- datadist(treat, num.diseases, age, cholesterol)
options(datadist='dd')
p <- Predict(g, age, treat="b")
s <- anova(g)
tx <- paste(capture.output(s), collapse='\n')
ggplot(p) + annotate('text', x=27, y=3.2, family='mono', label=tx,
                     hjust=0, vjust=1, size=1.5)

plot(s, margin=c('chisq', 'proportion chisq'))
# new plot - dot chart of chisq-d.f. with 2 other stats in right margin
# latex(s)                      # nice printout - creates anova.g.tex
options(datadist=NULL)


# Simulate data with from a given model, and display exactly which
# hypotheses are being tested


set.seed(123)
age <- rnorm(500, 50, 15)
treat <- factor(sample(c('a','b','c'), 500, TRUE))
bp  <- rnorm(500, 120, 10)
y   <- ifelse(treat=='a', (age-50)*.05, abs(age-50)*.08) + 3*(treat=='c') +
       pmax(bp, 100)*.09 + rnorm(500)
f   <- ols(y ~ treat*lsp(age,50) + rcs(bp,4))
print(names(coef(f)), quote=FALSE)
specs(f)
anova(f)
an <- anova(f)
options(digits=3)
print(an, 'subscripts')
print(an, 'dots')


an <- anova(f, test='Chisq', ss=FALSE)
# plot(0:1)                       # make some plot
# tab <- pantext(an, 1.2, .6, lattice=FALSE, fontfamily='Helvetica')
# create function to write table; usually omit fontfamily
# tab()                           # execute it; could do tab(cex=.65)
plot(an)                        # new plot - dot chart of chisq-d.f.
# Specify plot(an, trans=sqrt) to use a square root scale for this plot
# latex(an)                     # nice printout - creates anova.f.tex


## Example to save partial R^2 for all predictors, along with overall
## R^2, from two separate fits, and to combine them with ggplot2

require(ggplot2)
```

```
set.seed(1)
n <- 100
x1 <- runif(n)
x2 <- runif(n)
y  <- (x1-.5)^2 + x2 + runif(n)
group <- c(rep('a', n/2), rep('b', n/2))
A <- NULL
for(g in c('a','b')) {
    f <- ols(y ~ pol(x1,2) + pol(x2,2) + pol(x1,2) %ia% pol(x2,2),
             subset=group==g)
    a <- plot(anova(f),
              what='partial R2', pl=FALSE, rm.totals=FALSE, sort='none')
    a <- a[-grep('NONLINEAR', names(a))]
    d <- data.frame(group=g, Variable=factor(names(a), names(a)),
                    partialR2=unname(a))
    A <- rbind(A, d)
  }
ggplot(A, aes(x=partialR2, y=Variable)) + geom_point() +
       facet_wrap(~ group) + xlab(ex <- expression(partial~R^2)) +
       scale_y_discrete(limits=rev)
ggplot(A, aes(x=partialR2, y=Variable, color=group)) + geom_point() +
       xlab(ex <- expression(partial~R^2)) +
       scale_y_discrete(limits=rev)

# Suppose that a researcher wants to make a big deal about a variable
# because it has the highest adjusted chi-square.  We use the
# bootstrap to derive 0.95 confidence intervals for the ranks of all
# the effects in the model.  We use the plot method for anova, with
# pl=FALSE to suppress actual plotting of chi-square - d.f. for each
# bootstrap repetition.
# It is important to tell plot.anova.rms not to sort the results, or
# every bootstrap replication would have ranks of 1,2,3,... for the stats.

n <- 300
set.seed(1)
d <- data.frame(x1=runif(n), x2=runif(n),  x3=runif(n),
   x4=runif(n), x5=runif(n), x6=runif(n),  x7=runif(n),
   x8=runif(n), x9=runif(n), x10=runif(n), x11=runif(n),
   x12=runif(n))
d$y <- with(d, 1*x1 + 2*x2 + 3*x3 +  4*x4  + 5*x5 + 6*x6 +
               7*x7 + 8*x8 + 9*x9 + 10*x10 + 11*x11 +
              12*x12 + 9*rnorm(n))

f <- ols(y ~ x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12, data=d)
B <- 20   # actually use B=1000
ranks <- matrix(NA, nrow=B, ncol=12)
rankvars <- function(fit)
  rank(plot(anova(fit), sort='none', pl=FALSE))
Rank <- rankvars(f)
for(i in 1:B) {
  j <- sample(1:n, n, TRUE)
  bootfit <- update(f, data=d, subset=j)
  ranks[i,] <- rankvars(bootfit)
```

```
  }
lim <- t(apply(ranks, 2, quantile, probs=c(.025,.975)))
predictor <- factor(names(Rank), names(Rank))
w <- data.frame(predictor, Rank, lower=lim[,1], upper=lim[,2])
ggplot(w, aes(x=predictor, y=Rank)) + geom_point() + coord_flip() +
  scale_y_continuous(breaks=1:12) +
  geom_errorbar(aes(ymin=lim[,1], ymax=lim[,2]), width=0)
```

`as.data.frame.Ocens` *Convert ‘Ocens‘ Object to Data Frame to Facilitate Subset*

Description

Converts an ‘Ocens‘ object to a data frame so that subsetting will preserve all needed attributes

Usage

```
## S3 method for class 'Ocens'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	an ‘Ocens‘ object
`row.names`	optional vector of row names
`optional`	set to ‘TRUE‘ if needed
`...`	ignored

Value

data frame containing a 2-column integer matrix with attributes

Author(s)

Frank Harrell

bj *Buckley-James Multiple Regression Model*

Description

bj fits the Buckley-James distribution-free least squares multiple regression model to a possibly right-censored response variable. This model reduces to ordinary least squares if there is no censoring. By default, model fitting is done after taking logs of the response variable. bj uses the rms class for automatic anova, fastbw, validate, Function, nomogram, summary, plot, bootcov, and other functions. The bootcov function may be worth using with bj fits, as the properties of the Buckley-James covariance matrix estimator are not fully known for strange censoring patterns.

For the print method, format of output is controlled by the user previously running options(prType="lang") where lang is "plain" (the default), "latex", or "html". When using html with Quarto or RMarkdown, results='asis' need not be written in the chunk header.

The residuals.bj function exists mainly to compute residuals and to censor them (i.e., return them as Surv objects) just as the original failure time variable was censored. These residuals are useful for checking to see if the model also satisfies certain distributional assumptions. To get these residuals, the fit must have specified y=TRUE.

The bjplot function is a special plotting function for objects created by bj with x=TRUE, y=TRUE in effect. It produces three scatterplots for every covariate in the model: the first plots the original situation, where censored data are distingushed from non-censored data by a different plotting symbol. In the second plot, called a renovated plot, vertical lines show how censored data were changed by the procedure, and the third is equal to the second, but without vertical lines. Imputed data are again distinguished from the non-censored by a different symbol.

The validate method for bj validates the Somers' Dxy rank correlation between predicted and observed responses, accounting for censoring.

The primary fitting function for bj is bj.fit, which does not allow missing data and expects a full design matrix as input.

Usage

```
bj(formula, data=environment(formula), subset, na.action=na.delete,
   link="log", control, method='fit', x=FALSE, y=FALSE,
   time.inc)

## S3 method for class 'bj'
print(x, digits=4, long=FALSE, coefs=TRUE,
title="Buckley-James Censored Data Regression", ...)

## S3 method for class 'bj'
residuals(object, type=c("censored","censored.normalized"),...)

bjplot(fit, which=1:dim(X)[[2]])

## S3 method for class 'bj'
```

```
validate(fit, method="boot", B=40,
         bw=FALSE,rule="aic",type="residual",sls=.05,aics=0,
         force=NULL, estimates=TRUE, pr=FALSE,
 tol=1e-7, rel.tolerance=1e-3, maxiter=15, ...)

bj.fit(x, y, control)
```

Arguments

`formula` an S statistical model formula. Interactions up to third order are supported. The left hand side must be a `Surv` object.

`data`, `subset`, `na.action` the usual statistical model fitting arguments

`fit` a fit created by `bj`, required for all functions except `bj`.

`x` a design matrix with or without a first column of ones, to pass to `bj.fit`. All models will have an intercept. For `print.bj` is a result of `bj`. For `bj`, set `x=TRUE` to include the design matrix in the fit object.

`y` a `Surv` object to pass to `bj.fit` as the two-column response variable. Only right censoring is allowed, and there need not be any censoring. For `bj`, set `y` to `TRUE` to include the two-column response matrix, with the event/censoring indicator in the second column. The first column will be transformed according to `link`, and depending on `na.action`, rows with missing data in the predictors or the response will be deleted.

`link` set to, for example, `"log"` (the default) to model the log of the response, or `"identity"` to model the untransformed response.

`control` a list containing any or all of the following components: `iter.max` (maximum number of iterations allowed, default is 20), `eps` (convergence criterion: concergence is assumed when the ratio of sum of squared errors from one iteration to the next is between 1-`eps` and 1+`eps`), `trace` (set to `TRUE` to monitor iterations), `tol` (matrix singularity criterion, default is 1e-7), and 'max.cycle' (in case of nonconvergence the program looks for a cycle that repeats itself, default is 30).

`method` set to `"model.frame"` or `"model.matrix"` to return one of those objects rather than the model fit.

`time.inc` setting for default time spacing. Default is 30 if time variable has `units="Day"`, 1 otherwise, unless maximum follow-up time < 1. Then max time/10 is used as `time.inc`. If `time.inc` is not given and max time/default `time.inc` is > 25, `time.inc` is increased.

`digits` number of significant digits to print if not 4.

`long` set to `TRUE` to print the correlation matrix for parameter estimates

`coefs` specify `coefs=FALSE` to suppress printing the table of model coefficients, standard errors, etc. Specify `coefs=n` to print only the first `n` regression coefficients in the model.

`title` a character string title to be passed to `prModFit`

`object` the result of `bj`

type — type of residual desired. Default is censored unnormalized residuals, defined as link(Y) - linear.predictors, where the link function was usually the log function. You can specify `type="censored.normalized"` to divide the residuals by the estimate of `sigma`.

which — vector of integers or character strings naming elements of the design matrix (the names of the original predictors if they entered the model linearly) for which to have `bjplot` make plots of only the variables listed in `which` (names or numbers).

`B`, `bw`, `rule`, `sls`, `aics`, `force`, `estimates`, `pr`, `tol`, `rel.tolerance`, `maxiter` — see `predab.resample`

... — ignored for `print`; passed through to `predab.resample` for `validate`

Details

The program implements the algorithm as described in the original article by Buckley & James. Also, we have used the original Buckley & James prescription for computing variance/covariance estimator. This is based on non-censored observations only and does not have any theoretical justification, but has been shown in simulation studies to behave well. Our experience confirms this view. Convergence is rather slow with this method, so you may want to increase the number of iterations. Our experience shows that often, in particular with high censoring, 100 iterations is not too many. Sometimes the method will not converge, but will instead enter a loop of repeating values (this is due to the discrete nature of Kaplan and Meier estimator and usually happens with small sample sizes). The program will look for such a loop and return the average betas. It will also issue a warning message and give the size of the cycle (usually less than 6).

Value

`bj` returns a fit object with similar information to what `survreg`, `psm`, `cph` would store as well as what `rms` stores and `units` and `time.inc`. `residuals.bj` returns a `Surv` object. One of the components of the `fit` object produced by `bj` (and `bj.fit`) is a vector called `stats` which contains the following names elements: `"Obs"`, `"Events"`, `"d.f."`,`"error d.f."`,`"sigma"`,`"g"`. Here `sigma` is the estimate of the residual standard deviation. `g` is the g-index. If the link function is `"log"`, the g-index on the anti-log scale is also returned as `gr`.

Author(s)

Janez Stare
Department of Biomedical Informatics
Ljubljana University
Ljubljana, Slovenia
<janez.stare@mf.uni-lj.si>

Harald Heinzl
Department of Medical Computer Sciences
Vienna University
Vienna, Austria
<harald.heinzl@akh-wien.ac.at>

Frank Harrell
Department of Biostatistics

Vanderbilt University
<fh@fharrell.com>

References

Buckley JJ, James IR. Linear regression with censored data. Biometrika 1979; 66:429–36.

Miller RG, Halpern J. Regression with censored data. Biometrika 1982; 69: 521–31.

James IR, Smith PJ. Consistency results for linear regression with censored data. Ann Statist 1984; 12: 590–600.

Lai TL, Ying Z. Large sample theory of a modified Buckley-James estimator for regression analysis with censored data. Ann Statist 1991; 19: 1370–402.

Hillis SL. Residual plots for the censored data linear regression model. Stat in Med 1995; 14: 2023–2036.

Jin Z, Lin DY, Ying Z. On least-squares regression with censored data. Biometrika 2006; 93:147–161.

See Also

rms, psm, survreg, cph, Surv, na.delete, na.detail.response, datadist, rcorr.cens, GiniMd, prModFit, dxy.cens

Examples

```
require(survival)
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
ftime  <- 10*rexp(200)
stroke <- ifelse(ftime > 10, 0, 1)
ftime  <- pmin(ftime, 10)
units(ftime) <- "Month"
age <- rnorm(200, 70, 10)
hospital <- factor(sample(c('a','b'),200,TRUE))
dd <- datadist(age, hospital)
options(datadist="dd")

# Prior to rms 6.0 and R 4.0 the following worked with 5 knots
f <- bj(Surv(ftime, stroke) ~ rcs(age,3) + hospital, x=TRUE, y=TRUE)
# add link="identity" to use a censored normal regression model instead
# of a lognormal one
anova(f)
fastbw(f)
validate(f, B=15)
plot(Predict(f, age, hospital))
# needs datadist since no explicit age,hosp.
coef(f)               # look at regression coefficients
coef(psm(Surv(ftime, stroke) ~ rcs(age,3) + hospital, dist='lognormal'))
                      # compare with coefficients from likelihood-based
                      # log-normal regression model
                      # use dist='gau' not under R
```

```
r <- resid(f, 'censored.normalized')
survplot(npsurv(r ~ 1), conf='none')
                      # plot Kaplan-Meier estimate of
                      # survival function of standardized residuals
survplot(npsurv(r ~ cut2(age, g=2)), conf='none')
                      # may desire both strata to be n(0,1)
options(datadist=NULL)
```

---

bootBCa *BCa Bootstrap on Existing Bootstrap Replicates*

Description

This functions constructs an object resembling one produced by the `boot` package's `boot` function, and runs that package's `boot.ci` function to compute BCa and percentile confidence limits. `bootBCa` can provide separate confidence limits for a vector of statistics when `estimate` has length greater than 1. In that case, `estimates` must have the same number of columns as `estimate` has values.

Usage

```
bootBCa(estimate, estimates, type=c('percentile','bca','basic'),
               n, seed, conf.int = 0.95)
```

Arguments

`estimate`	original whole-sample estimate
`estimates`	vector of bootstrap estimates
`type`	type of confidence interval, defaulting to nonparametric percentile
`n`	original number of observations
`seed`	`.Random.seem` in effect before bootstrap estimates were run
`conf.int`	confidence level

Value

a 2-vector if `estimate` is of length 1, otherwise a matrix with 2 rows and number of columns equal to the length of `estimate`

Note

You can use `if(!exists('.Random.seed'))` `runif(1)` before running your bootstrap to make sure that `.Random.seed` will be available to `bootBCa`.

Author(s)

Frank Harrell

See Also

boot.ci

Examples

```
## Not run:
x1 <- runif(100); x2 <- runif(100); y <- sample(0:1, 100, TRUE)
f <- lrm(y ~ x1 + x2, x=TRUE, y=TRUE)
seed <- .Random.seed
b <- bootcov(f)
# Get estimated log odds at x1=.4, x2=.6
X <- cbind(c(1,1), x1=c(.4,2), x2=c(.6,3))
est <- X
ests <- t(X
bootBCa(est, ests, n=100, seed=seed)
bootBCa(est, ests, type='bca', n=100, seed=seed)
bootBCa(est, ests, type='basic', n=100, seed=seed)

## End(Not run)
```

---

bootcov *Bootstrap Covariance and Distribution for Regression Coefficients*

Description

bootcov computes a bootstrap estimate of the covariance matrix for a set of regression coefficients from ols, lrm, cph, psm, Rq, and any other fit where x=TRUE, y=TRUE was used to store the data used in making the original regression fit and where an appropriate fitter function is provided here. The estimates obtained are not conditional on the design matrix, but are instead unconditional estimates. For small sample sizes, this will make a difference as the unconditional variance estimates are larger. This function will also obtain bootstrap estimates corrected for cluster sampling (intra-cluster correlations) when a "working independence" model was used to fit data which were correlated within clusters. This is done by substituting cluster sampling with replacement for the usual simple sampling with replacement. bootcov has an option (coef.reps) that causes all of the regression coefficient estimates from all of the bootstrap re-samples to be saved, facilitating computation of nonparametric bootstrap confidence limits and plotting of the distributions of the coefficient estimates (using histograms and kernel smoothing estimates).

The loglik option facilitates the calculation of simultaneous confidence regions from quantities of interest that are functions of the regression coefficients, using the method of Tibshirani(1996). With Tibshirani's method, one computes the objective criterion (-2 log likelihood evaluated at the bootstrap estimate of β but with respect to the original design matrix and response vector) for the original fit as well as for all of the bootstrap fits. The confidence set of the regression coefficients is the set of all coefficients that are associated with objective function values that are less than or equal to say the 0.95 quantile of the vector of B + 1 objective function values. For the coefficients satisfying this condition, predicted values are computed at a user-specified design matrix X, and minima and maxima of these predicted values (over the qualifying bootstrap repetitions) are computed to derive the final simultaneous confidence band.

The bootplot function takes the output of bootcov and either plots a histogram and kernel density estimate of specified regression coefficients (or linear combinations of them through the use of a specified design matrix X), or a qqnorm plot of the quantities of interest to check for normality of the maximum likelihood estimates. bootplot draws vertical lines at specified quantiles of the bootstrap distribution, and returns these quantiles for possible printing by the user. Bootstrap estimates may optionally be transformed by a user-specified function fun before plotting.

The confplot function also uses the output of bootcov but to compute and optionally plot non-parametric bootstrap pointwise confidence limits or (by default) Tibshirani (1996) simultaneous confidence sets. A design matrix must be specified to allow confplot to compute quantities of interest such as predicted values across a range of values or differences in predicted values (plots of effects of changing one or more predictor variable values).

bootplot and confplot are actually generic functions, with the particular functions bootplot.bootcov and confplot.bootcov automatically invoked for bootcov objects.

A service function called histdensity is also provided (for use with bootplot). It runs hist and density on the same plot, using twice the number of classes than the default for hist, and 1.5 times the width than the default used by density.

A comprehensive example demonstrates the use of all of the functions.

When bootstrapping an ordinal model for a numeric Y (when ytarget is not specified), some original distinct Y values are not sampled so there will be fewer intercepts in the model. bootcov linearly interpolates and extrapolates to fill in the missing intercepts so that the intercepts are aligned over bootstrap samples. Also see the Hmisc ordGroupBoot function.

Usage

```
bootcov(fit, cluster, B=200, fitter,
        coef.reps=TRUE, loglik=FALSE,
        pr=FALSE, group=NULL, stat=NULL,
        seed=sample(10000, 1), ytarget=NULL, ...)


bootplot(obj, which=1 : ncol(Coef), X,
         conf.int=c(.9,.95,.99),
         what=c('density', 'qqnorm', 'box'),
         fun=function(x) x, labels., ...)


confplot(obj, X, against,
         method=c('simultaneous','pointwise'),
         conf.int=0.95, fun=function(x)x,
         add=FALSE, lty.conf=2, ...)


histdensity(y, xlab, nclass, width, mult.width=1, ...)
```

Arguments

fit	a fit object containing components x and y. For fits from cph, the "strata" attribute of the x component is used to obtain the vector of stratum codes.

obj	an object created by bootcov with coef.reps=TRUE.
X	a design matrix specified to confplot. See predict.rms or contrast.rms. For bootplot, X is optional.
y	a vector to pass to histdensity. NAs are ignored.
cluster	a variable indicating groupings. cluster may be any type of vector (factor, character, integer). Unique values of cluster indicate possibly correlated groupings of observations. Note the data used in the fit and stored in fit$x and fit$y may have had observations containing missing values deleted. It is assumed that if there were any NAs, an naresid function exists for the class of fit. This function restores NAs so that the rows of the design matrix coincide with cluster.
B	number of bootstrap repetitions. Default is 200.
fitter	the name of a function with arguments (x,y) that will fit bootstrap samples. Default is taken from the class of fit if it is ols, lrm, cph, psm, Rq.
coef.reps	set to TRUE if you want to store a matrix of all bootstrap regression coefficient estimates in the returned component boot.Coef.
loglik	set to TRUE to store -2 log likelihoods for each bootstrap model, evaluated against the original x and y data. The default is to do this when coef.reps is specified as TRUE. The use of loglik=TRUE assumes that an oos.loglik method exists for the type of model being analyzed, to calculate out-of-sample -2 log likelihoods (see rmsMisc). After the B -2 log likelihoods (stored in the element named boot.loglik in the returned fit object), the B+1 element is the -2 log likelihood for the original model fit.
pr	set to TRUE to print the current sample number to monitor progress.
group	a grouping variable used to stratify the sample upon bootstrapping. This allows one to handle k-sample problems, i.e., each bootstrap sample will be forced to select the same number of observations from each level of group as the number appearing in the original dataset. You may specify both group and cluster.
stat	a single character string specifying the name of a stats element produced by the fitting function to save over the bootstrap repetitions. The vector of saved statistics will be in the boot.stats part of the list returned by bootcov.
seed	random number seed for set.seed, defaults to a random integer between 1 and 10000; user should specify a constant for reproducibility
ytarget	when using orm, set ytarget=NA to save only the intercept that corresponds to the median Y. Set ytarget to a specific value (including a character value) to use a different target for the sole retained intercept.
which	one or more integers specifying which regression coefficients to plot for bootplot
conf.int	a vector (for bootplot, default is c(.9,.95,.99)) or scalar (for confplot, default is .95) confidence level.
what	for bootplot, specifies whether a density or a q-q plot is made, a ggplot2 is used to produce a box plot of all coefficients over the bootstrap reps
fun	for bootplot or confplot specifies a function used to translate the quantities of interest before analysis. A common choice is fun=exp to compute anti-logs, e.g., odds ratios.

labels.	a vector of labels for labeling the axes in plots produced by bootplot. Default is row names of X if there are any, or sequential integers.
...	For bootcov, extra arguments to pass to any of the fitting functions. For bootplot these are optional arguments passed to histdensity. Also may be optional arguments passed to plot by confplot or optional arguments passed to hist from histdensity, such as xlim and breaks. The argument probability=TRUE is always passed to hist.
against	For confplot, specifying against causes a plot to be made (or added to). The against variable is associated with rows of X and is used as the x-coordinates.
method	specifies whether "pointwise" or "simultaneous" confidence regions are derived by confplot. The default is simultaneous.
add	set to TRUE to add to an existing plot, for confplot
lty.conf	line type for plotting confidence bands in confplot. Default is 2 for dotted lines.
xlab	label for x-axis for histdensity. Default is label attribute or argument name if there is no label.
nclass	passed to hist if present
width	passed to density if present
mult.width	multiplier by which to adjust the default width passed to density. Default is 1.

Details

If the fit has a scale parameter (e.g., a fit from psm), the log of the individual bootstrap scale estimates are added to the vector of parameter estimates and and column and row for the log scale are added to the new covariance matrix (the old covariance matrix also has this row and column).

For Rq fits, the tau, method, and hs arguments are taken from the original fit.

Value

a new fit object with class of the original object and with the element orig.var added. orig.var is the covariance matrix of the original fit. Also, the original var component is replaced with the new bootstrap estimates. The component boot.coef is also added. This contains the mean bootstrap estimates of regression coefficients (with a log scale element added if applicable). boot.Coef is added if coef.reps=TRUE. boot.loglik is added if loglik=TRUE. If stat is specified an additional vector boot.stats will be contained in the returned object. B contains the number of successfully fitted bootstrap resamples. A component clusterInfo is added to contain elements name and n holding the name of the cluster variable and the number of clusters.

bootplot returns a (possible matrix) of quantities of interest and the requested quantiles of them. confplot returns three vectors: fitted, lower, and upper.

Side Effects

bootcov prints if pr=TRUE

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
<fh@fharrell.com>

Bill Pikounis
Biometrics Research Department
Merck Research Laboratories
https://billpikounis.com/wpb/

References

Feng Z, McLerran D, Grizzle J (1996): A comparison of statistical methods for clustered data analysis with Gaussian error. Stat in Med 15:1793–1806.

Tibshirani R, Knight K (1996): Model search and inference by bootstrap "bumping". Department of Statistics, University of Toronto. Technical report available from http://www-stat.stanford.edu/~tibs/. Presented at the Joint Statistical Meetings, Chicago, August 1996.

See Also

ordGroupBoot, robcov, sample, rms, lm.fit, lrm.fit, orm.fit, survival-internal, predab.resample, rmsMisc, Predict, gendata, contrast.rms, Predict, setPb, multiwayvcov::cluster.boot

Examples

```
set.seed(191)
x <- exp(rnorm(200))
logit <- 1 + x/2
y <- ifelse(runif(200) <= plogis(logit), 1, 0)
f <- lrm(y ~ pol(x,2), x=TRUE, y=TRUE)
g <- bootcov(f, B=50, pr=TRUE, seed=3)
anova(g)    # using bootstrap covariance estimates
fastbw(g)   # using bootstrap covariance estimates
beta <- g$boot.Coef[,1]
hist(beta, nclass=15)     #look at normality of parameter estimates
qqnorm(beta)
# bootplot would be better than these last two commands


# A dataset contains a variable number of observations per subject,
# and all observations are laid out in separate rows. The responses
# represent whether or not a given segment of the coronary arteries
# is occluded. Segments of arteries may not operate independently
# in the same patient.  We assume a "working independence model" to
# get estimates of the coefficients, i.e., that estimates assuming
# independence are reasonably efficient.  The job is then to get
# unbiased estimates of variances and covariances of these estimates.
```

```
set.seed(2)
n.subjects <- 30
ages <- rnorm(n.subjects, 50, 15)
sexes  <- factor(sample(c('female','male'), n.subjects, TRUE))
logit <- (ages-50)/5
prob <- plogis(logit)  # true prob not related to sex
id <- sample(1:n.subjects, 300, TRUE) # subjects sampled multiple times
table(table(id))  # frequencies of number of obs/subject
age <- ages[id]
sex <- sexes[id]
# In truth, observations within subject are independent:
y   <- ifelse(runif(300) <= prob[id], 1, 0)
f <- lrm(y ~ lsp(age,50)*sex, x=TRUE, y=TRUE)
g <- bootcov(f, id, B=50, seed=3)  # usually do B=200 or more
diag(g$var)/diag(f$var)
# add ,group=w to re-sample from within each level of w
anova(g)            # cluster-adjusted Wald statistics
# fastbw(g)         # cluster-adjusted backward elimination
plot(Predict(g, age=30:70, sex='female'))  # cluster-adjusted confidence bands


# Get design effects based on inflation of the variances when compared
# with bootstrap estimates which ignore clustering
g2 <- bootcov(f, B=50, seed=3)
diag(g$var)/diag(g2$var)


# Get design effects based on pooled tests of factors in model
anova(g2)[,1] / anova(g)[,1]


# Simulate binary data where there is a strong
# age x sex interaction with linear age effects
# for both sexes, but where not knowing that
# we fit a quadratic model.  Use the bootstrap
# to get bootstrap distributions of various
# effects, and to get pointwise and simultaneous
# confidence limits


set.seed(71)
n   <- 500
age <- rnorm(n, 50, 10)
sex <- factor(sample(c('female','male'), n, rep=TRUE))
L   <- ifelse(sex=='male', 0, .1*(age-50))
y   <- ifelse(runif(n)<=plogis(L), 1, 0)


f <- lrm(y ~ sex*pol(age,2), x=TRUE, y=TRUE)
b <- bootcov(f, B=50, loglik=TRUE, pr=TRUE, seed=3)   # better: B=500
```

```
par(mfrow=c(2,3))
# Assess normality of regression estimates
bootplot(b, which=1:6, what='qq')
# They appear somewhat non-normal


# Plot histograms and estimated densities
# for 6 coefficients
w <- bootplot(b, which=1:6)
# Print bootstrap quantiles
w$quantiles

# Show box plots for bootstrap reps for all coefficients
bootplot(b, what='box')


# Estimate regression function for females
# for a sequence of ages
ages <- seq(25, 75, length=100)
label(ages) <- 'Age'


# Plot fitted function and pointwise normal-
# theory confidence bands
par(mfrow=c(1,1))
p <- Predict(f, age=ages, sex='female')
plot(p)
# Save curve coordinates for later automatic
# labeling using labcurve in the Hmisc library
curves <- vector('list',8)
curves[[1]] <- with(p, list(x=age, y=lower))
curves[[2]] <- with(p, list(x=age, y=upper))


# Add pointwise normal-distribution confidence
# bands using unconditional variance-covariance
# matrix from the 500 bootstrap reps
p <- Predict(b, age=ages, sex='female')
curves[[3]] <- with(p, list(x=age, y=lower))
curves[[4]] <- with(p, list(x=age, y=upper))


dframe <- expand.grid(sex='female', age=ages)
X <- predict(f, dframe, type='x')  # Full design matrix


# Add pointwise bootstrap nonparametric
# confidence limits
p <- confplot(b, X=X, against=ages, method='pointwise',
              add=TRUE, lty.conf=4)
curves[[5]] <- list(x=ages, y=p$lower)
curves[[6]] <- list(x=ages, y=p$upper)
```

```
# Add simultaneous bootstrap confidence band
p <- confplot(b, X=X, against=ages, add=TRUE, lty.conf=5)
curves[[7]] <- list(x=ages, y=p$lower)
curves[[8]] <- list(x=ages, y=p$upper)
lab <- c('a','a','b','b','c','c','d','d')
labcurve(curves, lab, pl=TRUE)


# Now get bootstrap simultaneous confidence set for
# female:male odds ratios for a variety of ages


dframe <- expand.grid(age=ages, sex=c('female','male'))
X <- predict(f, dframe, type='x')  # design matrix
f.minus.m <- X[1:100,] - X[101:200,]
# First 100 rows are for females.  By subtracting
# design matrices are able to get Xf*Beta - Xm*Beta
# = (Xf - Xm)*Beta


confplot(b, X=f.minus.m, against=ages,
         method='pointwise', ylab='F:M Log Odds Ratio')
confplot(b, X=f.minus.m, against=ages,
         lty.conf=3, add=TRUE)


# contrast.rms makes it easier to compute the design matrix for use
# in bootstrapping contrasts:


f.minus.m <- contrast(f, list(sex='female',age=ages),
                         list(sex='male',  age=ages))$X
confplot(b, X=f.minus.m)


# For a quadratic binary logistic regression model use bootstrap
# bumping to estimate coefficients under a monotonicity constraint
set.seed(177)
n <- 400
x <- runif(n)
logit <- 3*(x^2-1)
y <- rbinom(n, size=1, prob=plogis(logit))
f <- lrm(y ~ pol(x,2), x=TRUE, y=TRUE)
k <- coef(f)
k
vertex <- -k[2]/(2*k[3])
vertex


# Outside [0,1] so fit satisfies monotonicity constraint within
# x in [0,1], i.e., original fit is the constrained MLE
```

```
g <- bootcov(f, B=50, coef.reps=TRUE, loglik=TRUE, seed=3)
bootcoef <- g$boot.Coef    # 100x3 matrix
vertex <- -bootcoef[,2]/(2*bootcoef[,3])
table(cut2(vertex, c(0,1)))
mono <- !(vertex >= 0 & vertex <= 1)
mean(mono)    # estimate of Prob{monotonicity in [0,1]}


var(bootcoef)   # var-cov matrix for unconstrained estimates
var(bootcoef[mono,])   # for constrained estimates


# Find second-best vector of coefficient estimates, i.e., best
# from among bootstrap estimates
g$boot.Coef[order(g$boot.loglik[-length(g$boot.loglik)])[1],]
# Note closeness to MLE

## Not run:
# Get the bootstrap distribution of the difference in two ROC areas for
# two binary logistic models fitted on the same dataset.  This analysis
# does not adjust for the bias ROC area (C-index) due to overfitting.
# The same random number seed is used in two runs to enforce pairing.

set.seed(17)
x1 <- rnorm(100)
x2 <- rnorm(100)
y <- sample(0:1, 100, TRUE)
f <- lrm(y ~ x1, x=TRUE, y=TRUE)
g <- lrm(y ~ x1 + x2, x=TRUE, y=TRUE)
f <- bootcov(f, stat='C', seed=4)
g <- bootcov(g, stat='C', seed=4)
dif <- g$boot.stats - f$boot.stats
hist(dif)
quantile(dif, c(.025,.25,.5,.75,.975))
# Compute a z-test statistic.  Note that comparing ROC areas is far less
# powerful than likelihood or Brier score-based methods
z <- (g$stats['C'] - f$stats['C'])/sd(dif)
names(z) <- NULL
c(z=z, P=2*pnorm(-abs(z)))

# For an ordinal y with some distinct values of y not very popular, let
# bootcov use linear extrapolation to fill in intercepts for non-sampled levels

f <- orm(y ~ x1 + x2, x=TRUE, y=TRUE)
bootcov(f, B=200)

# Instead of filling in missing intercepts, perform minimum binning so that
# there is a 0.9999 probability that all distinct Y values will be represented
# in bootstrap samples
y <- ordGroupBoot(y)
f <- orm(y ~ x1 + x2, x=TRUE, y=TRUE)
bootcov(f, B=200)
```

```
# Instead just keep one intercept for all bootstrap fits - the intercept
# that pertains to y=10

bootcov(f, B=200, ytarget=10)   # use ytarget=NA for the median

## End(Not run)
```

---

bplot *3-D Plots Showing Effects of Two Continuous Predictors in a Regression Model Fit*

Description

Uses lattice graphics and the output from `Predict` to plot image, contour, or perspective plots showing the simultaneous effects of two continuous predictor variables. Unless `formula` is provided, the x-axis is constructed from the first variable listed in the call to `Predict` and the y-axis variable comes from the second.

The `perimeter` function is used to generate the boundary of data to plot when a 3-d plot is made. It finds the area where there are sufficient data to generate believable interaction fits.

Usage

```
bplot(x, formula, lfun=lattice::levelplot, xlab, ylab, zlab,
      adj.subtitle=!info$ref.zero, cex.adj=.75, cex.lab=1,
      perim, showperim=FALSE,
      zlim=range(yhat, na.rm=TRUE), scales=list(arrows=FALSE),
      xlabrot, ylabrot, zlabrot=90, ...)

perimeter(x, y, xinc=diff(range(x))/10, n=10, lowess.=TRUE)
```

Arguments

`x`	for `bplot`, an object created by `Predict` for which two or more numeric predictors varied. For `perim` is the first variable of a pair of predictors forming a 3-d plot.
`formula`	a formula of the form `f(yhat) ~ x*y` optionally followed by |a*b*c which are 1-3 paneling variables that were specified to `Predict`. `f` can represent any R function of a vector that produces a vector. If the left hand side of the formula is omitted, `yhat` will be inserted. If `formula` is omitted, it will be inferred from the first two variables that varied in the call to `Predict`.
`lfun`	a high-level lattice plotting function that takes formulas of the form `z ~ x*y`. The default is an image plot (`levelplot`). Other common choices are `wireframe` for perspective plot or `contourplot` for a contour plot.
`xlab`	Character string label for x-axis. Default is given by `Predict`.
`ylab`	Character string abel for y-axis

zlab	Character string z-axis label for perspective (wireframe) plots. Default comes from Predict. zlab will often be specified if fun was specified to Predict.
adj.subtitle	Set to FALSE to suppress subtitling the graph with the list of settings of non-graphed adjustment values. Default is TRUE if there are non-plotted adjustment variables and ref.zero was not used.
cex.adj	cex parameter for size of adjustment settings in subtitles. Default is 0.75
cex.lab	cex parameter for axis labels. Default is 1.
perim	names a matrix created by perimeter when used for 3-d plots of two continuous predictors. When the combination of variables is outside the range in perim, that section of the plot is suppressed. If perim is omitted, 3-d plotting will use the marginal distributions of the two predictors to determine the plotting region, when the grid is not specified explicitly in variables. When instead a series of curves is being plotted, perim specifies a function having two arguments. The first is the vector of values of the first variable that is about to be plotted on the x-axis. The second argument is the single value of the variable representing different curves, for the current curve being plotted. The function's returned value must be a logical vector whose length is the same as that of the first argument, with values TRUE if the corresponding point should be plotted for the current curve, FALSE otherwise. See one of the latter examples.
showperim	set to TRUE if perim is specified and you want to show the actual perimeter used.
zlim	Controls the range for plotting in the z-axis if there is one. Computed by default.
scales	see wireframe
xlabrot	rotation angle for the x-axis. Default is 30 for wireframe and 0 otherwise.
ylabrot	rotation angle for the y-axis. Default is -40 for wireframe, 90 for contourplot or levelplot, and 0 otherwise.
zlabrot	rotation angle for z-axis rotation for wireframe plots
...	other arguments to pass to the lattice function
y	second variable of the pair for perim. If omitted, x is assumed to be a list with both x and y components.
xinc	increment in x over which to examine the density of y in perimeter
n	within intervals of x for perimeter, takes the informative range of y to be the nth smallest to the nth largest values of y. If there aren't at least $2n$ y values in the x interval, no y ranges are used for that interval.
lowess.	set to FALSE to not have lowess smooth the data perimeters

Details

perimeter is a kind of generalization of datadist for 2 continuous variables. First, the n smallest and largest x values are determined. These form the lowest and highest possible xs to display. Then x is grouped into intervals bounded by these two numbers, with the interval widths defined by xinc. Within each interval, y is sorted and the nth smallest and largest y are taken as the interval containing sufficient data density to plot interaction surfaces. The interval is ignored when there are insufficient y values. When the data are being readied for persp, bplot uses the approx function to do linear interpolation of the y-boundaries as a function of the x values actually used in forming the grid (the values of the first variable specified to Predict). To make the perimeter smooth, specify lowess.=TRUE to perimeter.

Value

perimeter returns a matrix of class perimeter. This outline can be conveniently plotted by lines.perimeter.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

datadist, Predict, rms, rmsMisc, levelplot, contourplot, wireframe

Examples

```
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
label(age)            <- 'Age'      # label is in Hmisc
label(cholesterol)    <- 'Total Cholesterol'
label(blood.pressure) <- 'Systolic Blood Pressure'
label(sex)            <- 'Sex'
units(cholesterol)    <- 'mg/dl'   # uses units.default in Hmisc
units(blood.pressure) <- 'mmHg'

# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
  (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male'))
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)

ddist <- datadist(age, blood.pressure, cholesterol, sex)
options(datadist='ddist')

fit <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)),
               x=TRUE, y=TRUE)
p <- Predict(fit, age, cholesterol, sex, np=50) # vary sex last
require(lattice)
bplot(p)                 # image plot for age, cholesterol with color
                         # coming from yhat; use default ranges for
                         # both continuous predictors; two panels (for sex)
bplot(p, lfun=wireframe) # same as bplot(p,,wireframe)
# View from different angle, change y label orientation accordingly
# Default is z=40, x=-60
bplot(p,, wireframe, screen=list(z=40, x=-75), ylabrot=-25)
bplot(p,, contourplot)   # contour plot
bounds  <- perimeter(age, cholesterol, lowess=TRUE)
plot(age, cholesterol)     # show bivariate data density and perimeter
```

```
lines(bounds[,c('x','ymin')]); lines(bounds[,c('x','ymax')])
p <- Predict(fit, age, cholesterol)  # use only one sex
bplot(p, perim=bounds)   # draws image() plot
                         # don't show estimates where data are sparse
                         # doesn't make sense here since vars don't interact
bplot(p, plogis(yhat) ~ age*cholesterol) # Probability scale
options(datadist=NULL)
```

---

calibrate *Resampling Model Calibration*

Description

Uses bootstrapping or cross-validation to get bias-corrected (overfitting- corrected) estimates of predicted vs. observed values based on subsetting predictions into intervals or better on nonparametric or adaptive parametric smoothers. There are calibration functions for Cox (`cph`), parametric survival models (`psm`), binary and ordinal logistic models (`lrm`, `orm`) and ordinary least squares (`ols`). For survival models and `orm`, "predicted" means predicted survival probability at a single time point, and "observed" refers to the corresponding Kaplan-Meier survival estimate, stratifying on intervals of predicted survival, or, the predicted survival probability as a function of transformed predicted survival probability using the flexible hazard regression approach or for `orm` and probably better, smoothed overlapping moving Kaplan-Meier estimates (see the `val.surv.args` argument and `val.surv` function for details). Nonparametric calibration curves are estimated over a regular sequence of predicted values. The fit must have specified `x=TRUE, y=TRUE`.

See `predab.resample` for information about confidence limits. Confidence limits for bootstrap overfitting-corrected calibration curves are not computed for `psm` fits. This is because of `calibrate.psm` averages over multiple bootstrap loops. This can probably be changed.

The `print` and `plot` methods print the mean absolute error in predictions, the mean squared error, and the 0.9 quantile of the absolute error. Here, error refers to the difference between the predicted values and the corresponding bias-corrected calibrated values.

Below, `calibrate.default` is for the `ols` and `lrm`.

Usage

```
calibrate(fit, ...)
## Default S3 method:
calibrate(fit, predy,
  method=c("boot","crossvalidation",".632","randomization"),
  B=40, bw=FALSE, rule=c("aic","p"),
  type=c("residual","individual"),
  sls=.05, aics=0, force=NULL, estimates=TRUE, pr=FALSE, kint,
  smoother="lowess", digits=NULL, ...)
## S3 method for class 'cph'
calibrate(fit, cmethod=c('hare', 'KM'),
  method="boot", u, m=150, pred, cuts, B=40,
  bw=FALSE, rule="aic", type="residual", sls=0.05, aics=0, force=NULL,
```

```
  estimates=TRUE,
  pr=FALSE, what="observed-predicted", tol=1e-12, maxdim=5, ...)
## S3 method for class 'orm'
calibrate(fit,
                        method="boot", u, m=150, pred, B=40,
                        bw=FALSE, rule="aic",
                        type="residual", sls=.05, aics=0, force=NULL,
                        estimates=TRUE, pr=FALSE, what="observed-predicted",
                        val.surv.args=list(method='smoothkm', eps=30),
                        ...)
## S3 method for class 'psm'
calibrate(fit, cmethod=c('hare', 'KM'),
  method="boot", u, m=150, pred, cuts, B=40,
  bw=FALSE,rule="aic",
  type="residual", sls=.05, aics=0, force=NULL, estimates=TRUE,
  pr=FALSE, what="observed-predicted", tol=1e-12, maxiter=15,
  rel.tolerance=1e-5, maxdim=5, ...)

## S3 method for class 'calibrate'
print(x, B=Inf, ...)
## S3 method for class 'calibrate.default'
print(x, B=Inf, ...)

## S3 method for class 'calibrate'
plot(x, xlab, ylab, subtitles=TRUE, conf.int=TRUE,
 cex.subtitles=.75, riskdist=TRUE, add=FALSE,
 scat1d.opts=list(nhistSpike=200), par.corrected=NULL, ...)

## S3 method for class 'calibrate.default'
plot(x, xlab, ylab, xlim, ylim,
  legend=TRUE, subtitles=TRUE, cex.subtitles=.75, riskdist=TRUE,
  scat1d.opts=list(nhistSpike=200), ...)
```

Arguments

`fit` a fit from `ols`, `lrm`, `cph` or `psm`

`x` an object created by `calibrate`

`method`, `B`, `bw`, `rule`, `type`, `sls`, `aics`, `force`, `estimates`
: see `validate`. For `print.calibrate`, `B` is an upper limit on the number of resamples for which information is printed about which variables were selected in each model re-fit. Specify zero to suppress printing. Default is to print all re-samples.

`cmethod` method for validating survival predictions using right-censored data. The default is `cmethod='hare'` to use the `hare` function in the `polspline` package. Specify `cmethod='KM'` to use less precision stratified Kaplan-Meier estimates. If the `polspline` package is not available, the procedure reverts to `cmethod='KM'`.

`u` the time point for which to validate predictions for survival models. For `cph` fits, you must have specified `surv=TRUE`, `time.inc=u`, where `u` is the constant

specifying the time to predict.

`m` group predicted u-time units survival into intervals containing `m` subjects on the average (for survival models only)

`pred` vector of predicted survival probabilities at which to evaluate the calibration curve. By default, the low and high prediction values from `datadist` are used, which for large sample size is the 10th smallest to the 10th largest predicted probability.

`cuts` actual cut points for predicted survival probabilities. You may specify only one of `m` and `cuts` (for survival models only)

`pr` set to `TRUE` to print intermediate results for each re-sample

`what` The default is `"observed-predicted"`, meaning to estimate optimism in this difference. This is preferred as it accounts for skewed distributions of predicted probabilities in outer intervals. You can also specify `"observed"`. This argument applies to survival models only.

`tol` criterion for matrix singularity (default is `1e-12`)

`maxdim` see `hare`

`maxiter` for `psm`, this is passed to `survreg.control` (default is 15 iterations)

`rel.tolerance` parameter passed to `survreg.control` for `psm` (default is 1e-5).

`predy` a scalar or vector of predicted values to calibrate (for `lrm`, `ols`). Default is 50 equally spaced points between the 5th smallest and the 5th largest predicted values. For `lrm` the predicted values are probabilities (see `kint`).

`kint` For an ordinal logistic model the default predicted probability that $Y \geq$ the middle level. Specify `kint` to specify the intercept to use, e.g., `kint=2` means to calibrate $Prob(Y \geq b)$, where b is the second level of Y.

`val.surv.args` a list containing arguments to send to `val.surv` when running `calibrate.orm`. By default smoothed overlapping windows of Kaplan-Meier estimates are used for `orm`. The `val.surv.args` argument is especially useful for specifying bandwidths and the `movStats` `eps` argument.

`smoother` a function in two variables which produces x- and y-coordinates by smoothing the input `y`. The default is to use `lowess(x, y, iter=0)`.

`digits` If specified, predicted values are rounded to `digits` digits before passing to the smoother. Occasionally, large predicted values on the logit scale will lead to predicted probabilities very near 1 that should be treated as 1, and the `round` function will fix that. Applies to `calibrate.default`.

`...` other arguments to pass to `predab.resample`, such as `conf.int`, `group`, `cluster`, and `subset`. Also, other arguments for `plot`.

`xlab` defaults to "Predicted x-units Survival" or to a suitable label for other models

`ylab` defaults to "Fraction Surviving x-units" or to a suitable label for other models

`xlim`, `ylim` 2-vectors specifying x- and y-axis limits, if not using defaults

`subtitles` set to `FALSE` to suppress subtitles in plot describing method and for `lrm` and `ols` the mean absolute error and original sample size

`conf.int` set to `FALSE` to suppress plotting 0.95 confidence intervals for Kaplan-Meier estimates

cex.subtitles	character size for plotting subtitles
riskdist	set to FALSE to suppress the distribution of predicted risks (survival probabilities) from being plotted
add	set to TRUE to add the calibration plot to an existing plot
scat1d.opts	a list specifying options to send to scat1d if riskdist=TRUE. See scat1d.
par.corrected	a list specifying graphics parameters col, lty, lwd, pch to be used in drawing overfitting-corrected estimates. Default is col="blue", lty=1, lwd=1, pch=4.
legend	set to FALSE to suppress legends (for lrm, ols only) on the calibration plot, or specify a list with elements x and y containing the coordinates of the upper left corner of the legend. By default, a legend will be drawn in the lower right 1/16th of the plot.

Details

If the fit was created using penalized maximum likelihood estimation, the same penalty and penalty.scale parameters are used during validation.

See https://www.fharrell.com/post/bootcal/ for simulations of the accuracy of various smoothers for binary logistic model calibration, as well as simulations of confidence interval coverage.

Value

matrix specifying mean predicted survival in each interval, the corresponding estimated bias-corrected Kaplan-Meier estimates, number of subjects, and other statistics. For linear and logistic models, the matrix instead has rows corresponding to the prediction points, and the vector of predicted values being validated is returned as an attribute. The returned object has class "calibrate" or "calibrate.default". plot.calibrate.default invisibly returns the vector of estimated prediction errors corresponding to the dataset used to fit the model.

Side Effects

prints, and stores an object pred.obs or .orig.cal

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

validate, predab.resample, groupkm, errbar, scat1d, cph, psm, lowess,fit.mult.impute, processMI, val.surv, orm, movStats

Examples

```
require(survival)
set.seed(1)
n <- 200
d.time <- rexp(n)
x1 <- runif(n)
x2 <- factor(sample(c('a', 'b', 'c'), n, TRUE))
f <- cph(Surv(d.time) ~ pol(x1,2) * x2, x=TRUE, y=TRUE, surv=TRUE, time.inc=1.5)
#or f <- psm(S ~ \dots)
pa <- requireNamespace('polspline')
if(pa) {
 cal <- calibrate(f, u=1.5, B=20)  # cmethod='hare'
 plot(cal)
}
cal <- calibrate(f, u=1.5, cmethod='KM', m=50, B=20)  # usually B=200 or 300
plot(cal, add=pa)

set.seed(1)
y <- sample(0:2, n, TRUE)
x1 <- runif(n)
x2 <- runif(n)
x3 <- runif(n)
x4 <- runif(n)
f <- lrm(y ~ x1 + x2 + x3 * x4, x=TRUE, y=TRUE)
cal <- calibrate(f, kint=2, predy=seq(.2, .8, length=60),
                 group=y)
# group= does k-sample validation: make resamples have same
# numbers of subjects in each level of y as original sample

plot(cal)
#See the example for the validate function for a method of validating
#continuation ratio ordinal logistic models.  You can do the same
#thing for calibrate
```

`contrast.rms` *General Contrasts of Regression Coefficients*

Description

This function computes one or more contrasts of the estimated regression coefficients in a fit from one of the functions in rms, along with standard errors, confidence limits, t or Z statistics, P-values. General contrasts are handled by obtaining the design matrix for two sets of predictor settings (`a`, `b`) and subtracting the corresponding rows of the two design matrics to obtain a new contrast design matrix for testing the `a - b` differences. This allows for quite general contrasts (e.g., estimated differences in means between a 30 year old female and a 40 year old male). This can also be used to obtain a series of contrasts in the presence of interactions (e.g., female:male log odds ratios for several ages when the model contains age by sex interaction). Another use of `contrast` is to obtain center-weighted (Type III test) and subject-weighted (Type II test) estimates in a model containing

treatment by center interactions. For the latter case, you can specify `type="average"` and an optional `weights` vector to average the within-center treatment contrasts. The design contrast matrix computed by `contrast.rms` can be used by other functions.

When the model was fitted by a Bayesian function such as `blrm`, highest posterior density intervals for contrasts are computed instead, along with the posterior probability that the contrast is positive. `posterior.summary` specifies whether posterior mean/median/mode is to be used for contrast point estimates.

`contrast.rms` also allows one to specify four settings to contrast, yielding contrasts that are double differences - the difference between the first two settings (`a - b`) and the last two (`a2 - b2`). This allows assessment of interactions.

If `usebootcoef=TRUE`, the fit was run through `bootcov`, and `conf.type="individual"`, the confidence intervals are bootstrap nonparametric percentile confidence intervals, basic bootstrap, or BCa intervals, obtained on contrasts evaluated on all bootstrap samples.

By omitting the `b` argument, `contrast` can be used to obtain an average or weighted average of a series of predicted values, along with a confidence interval for this average. This can be useful for "unconditioning" on one of the predictors (see the next to last example).

Specifying `type="joint"`, and specifying at least as many contrasts as needed to span the space of a complex test, one can make multiple degree of freedom tests flexibly and simply. Redundant contrasts will be ignored in the joint test. See the examples below. These include an example of an "incomplete interaction test" involving only two of three levels of a categorical variable (the test also tests the main effect).

When more than one contrast is computed, the list created by `contrast.rms` is suitable for plotting (with error bars or bands) with `xYplot` or `Dotplot` (see the last example before the `type="joint"` examples).

When `fit` is the result of a Bayesian model fit and `fun` is specified, `contrast.rms` operates altogether differently. `a` and `b` must both be specified and `a2, b2` not specified. `fun` is evaluated on the estimates separately on `a` and `b` and the subtraction is deferred. So even in the absence of interactions, when `fun` is nonlinear, the settings of factors (predictors) will not cancel out and estimates of differences will be covariate-specific (unless there are no covariates in the model besides the one being varied to get from `a` to `b`).

That the the use of offsets to compute profile confidence intervals prevents this function from working with certain models that use offsets for other purposes, e.g., Poisson models with offsets to account for population size.

Usage

```
contrast(fit, ...)
## S3 method for class 'rms'
contrast(fit, a, b, a2, b2, ycut=NULL, cnames=NULL,
         fun=NULL, funint=TRUE,
         type=c("individual", "average", "joint"),
         conf.type=c("individual","simultaneous","profile"), usebootcoef=TRUE,
         boot.type=c("percentile","bca","basic"),
         posterior.summary=c('mean', 'median', 'mode'),
         weights="equal", conf.int=0.95, tol=1e-7, expand=TRUE,
         se_factor=4, plot_profile=FALSE, ...)
```

```
## S3 method for class 'contrast.rms'
print(x, X=FALSE,
       fun=function(u)u, jointonly=FALSE, prob=0.95, ...)
```

Arguments

fit	a fit of class "rms"
a	a list containing settings for all predictors that you do not wish to set to default (adjust-to) values. Usually you will specify two variables in this list, one set to a constant and one to a sequence of values, to obtain contrasts for the sequence of values of an interacting factor. The gendata function will generate the necessary combinations and default values for unspecified predictors, depending on the expand argument.
b	another list that generates the same number of observations as a, unless one of the two lists generates only one observation. In that case, the design matrix generated from the shorter list will have its rows replicated so that the contrasts assess several differences against the one set of predictor values. This is useful for comparing multiple treatments with control, for example. If b is missing, the design matrix generated from a is analyzed alone.
a2	an optional third list of settings of predictors
b2	an optional fourth list of settings of predictors. Mandatory if a2 is given.
ycut	used of the fit is a constrained partial proportional odds model fit, to specify the single value or vector of values (corresponding to the multiple contrasts) of the response variable to use in forming contrasts. When there is non-proportional odds, odds ratios will vary over levels of the response variable. When there are multiple contrasts and only one value is given for ycut, that value will be propagated to all contrasts. To show the effect of non-proportional odds, let ycut vary.
cnames	vector of character strings naming the contrasts when type!="average". Usually cnames is not necessary as contrast.rms tries to name the contrasts by examining which predictors are varying consistently in the two lists. cnames will be needed when you contrast "non-comparable" settings, e.g., you compare list(treat="drug", age=c(20,30)) with list(treat="placebo"), age=c(40,50))
fun	a function to evaluate on the linear predictor for each of a and b. Applies to Bayesian model fits. Also, a function to transform the contrast, SE, and lower and upper confidence limits before printing. For example, specify fun=exp to anti-log them for logistic models.
type	set type="average" to average the individual contrasts (e.g., to obtain a Type II or III contrast). Set type="joint" to jointly test all non-redundant contrasts with a multiple degree of freedom test and no averaging.
conf.type	The default type of confidence interval computed for a given individual (1 d.f.) contrast is a pointwise confidence interval. Set conf.type="simultaneous" to use the multcomp package's glht and confint functions to compute confidence intervals with simultaneous (family-wise) coverage, thus adjusting for multiple comparisons. Note that individual P-values are not adjusted for multiplicity.

usebootcoef — If fit was the result of bootcov but you want to use the bootstrap covariance matrix instead of the nonparametric percentile, basic, or BCa method for confidence intervals (which uses all the bootstrap coefficients), specify usebootcoef=FALSE.

boot.type — set to 'bca' to compute BCa confidence limits or 'basic' to use the basic bootstrap. The default is to compute percentile intervals

posterior.summary — By default the posterior mean is used. Specify posterior.summary='median' to instead use the posterior median and likewise posterior.summary='mode'. Unlike other functions, contrast.rms does not default to 'mode' because point estimates come from contrasts and not the original model coefficients point estimates.

weights — a numeric vector, used when type="average", to obtain weighted contrasts

conf.int — confidence level for confidence intervals for the contrasts (HPD interval probability for Bayesian analyses)

tol — tolerance for qr function for determining which contrasts are redundant, and for inverting the covariance matrix involved in a joint test. This should be larger than the usual tolerance chosen when just inverting a matrix.

expand — set to FALSE to have gendata not generate all possible combinations of predictor settings. This is useful when getting contrasts over irregular predictor settings.

se_factor — multiplier for a contrast's standard error used for root finding of the profile likelihood confidence limits when conf.type='profile'. The search is over the maximum likelihood estimate plus or minus se_factor times the standard error. This approach will fail when the Hauck-Donner effect is in play, because the standard error blows up when regression coefficients are estimating infinity.

plot_profile — when conf.type='profile' specify plot_profile to plot the change in deviance from the full model as a function of the contrast estimate, separately by each row of the contrast matrix. The contrast estimate varies from the maximum likelihood estimate plus or minus se_factor times the standard error, with a regular grid of 50 points.

... — passed to print for main output. A useful thing to pass is digits=4. Used also to pass convergence criteria arguments to fitting functions when conf.type is "profile".

x — result of contrast

X — set X=TRUE to print design matrix used in computing the contrasts (or the average contrast)

funint — set to FALSE if fun is not a function such as the result of Mean, Quantile, or ExProb that contains an intercepts argument

jointonly — set to FALSE to omit printing of individual contrasts

prob — highest posterior density interval probability when the fit was Bayesian and fun was specified to contrast.rms

Value

a list of class "contrast.rms" containing the elements Contrast, SE, Z, var, df.residual Lower, Upper, Pvalue, X, cnames, redundant, which denote the contrast estimates, standard errors, Z or

t-statistics, variance matrix, residual degrees of freedom (this is NULL if the model was not ols), lower and upper confidence limits, 2-sided P-value, design matrix, contrast names (or NULL), and a logical vector denoting which contrasts are redundant with the other contrasts. If there are any redundant contrasts, when the results of contrast are printed, and asterisk is printed at the start of the corresponding lines. The object also contains ctype indicating what method was used for compute confidence intervals.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
fh@fharrell.com

See Also

Predict, gendata, bootcov, summary.rms, anova.rms,

Examples

```
require(ggplot2)
set.seed(1)
age <- rnorm(200,40,12)
sex <- factor(sample(c('female','male'),200,TRUE))
logit <- (sex=='male') + (age-40)/5
y <- ifelse(runif(200) <= plogis(logit), 1, 0)
f <- lrm(y ~ pol(age,2)*sex)
anova(f)
# Compare a 30 year old female to a 40 year old male
# (with or without age x sex interaction in the model)
contrast(f, list(sex='female', age=30), list(sex='male', age=40))
# Test for interaction between age and sex, duplicating anova
contrast(f, list(sex='female', age=30),
            list(sex='male',   age=30),
            list(sex='female', age=c(40,50)),
            list(sex='male',   age=c(40,50)), type='joint')
# Duplicate overall sex effect in anova with 3 d.f.
contrast(f, list(sex='female', age=c(30,40,50)),
            list(sex='male',   age=c(30,40,50)), type='joint')
# For females get an array of odds ratios against age=40
k <- contrast(f, list(sex='female', age=30:50),
                 list(sex='female', age=40))
print(k, fun=exp)
# Plot odds ratios with pointwise 0.95 confidence bands using log scale
k <- as.data.frame(k[c('Contrast','Lower','Upper')])
ggplot(k, aes(x=30:50, y=exp(Contrast))) + geom_line() +
   geom_ribbon(aes(ymin=exp(Lower), ymax=exp(Upper)),
               alpha=0.15, linetype=0) +
   scale_y_continuous(trans='log10', n.breaks=10,
               minor_breaks=c(seq(0.1, 1, by=.1), seq(1, 10, by=.5))) +
  xlab('Age') + ylab('OR against age 40')
```

```
# For an ordinal model with 3 variables (x1 is quadratic, x2 & x3 linear)
# Get a 1 d.f. likelihood ratio (LR) test for x1=1 vs x1=0.25
# For the other variables get contrasts and LR tests that are the
# ordinary ones for their original coefficients.
# Get 0.95 profile likelihood confidence intervals for the x1 contrast
# and for the x2 and x3 coefficients
set.seed(7)
x1 <- runif(50)
x2 <- runif(50)
x3 <- runif(50)
dd <- datadist(x1, x2, x3); options(datadist='dd')
y <- x1 + runif(50)   # need x=TRUE,y=TRUE for profile likelihood
f <- orm(y ~ pol(x1, 2) + x2 + x3, x=TRUE, y=TRUE)
a <- list(x1=c(   1,0,0), x2=c(0,1,0), x3=c(0,0,1))
b <- list(x1=c(0.25,0,0), x2=c(0,0,0), x3=c(0,0,0))
k <- contrast(f, a, b, expand=FALSE)      # Wald intervals and tests
k; k$X[1,]
summary(f, x1=c(.25, 1), x2=0:1, x3=0:1)  # Wald intervals
anova(f, test='LR')                       # LR tests
contrast(f, a, b, expand=FALSE, conf.type='profile', plot_profile=TRUE)
options(datadist=NULL)


# For a model containing two treatments, centers, and treatment
# x center interaction, get 0.95 confidence intervals separately
# by center
center <- factor(sample(letters[1 : 8], 500, TRUE))
treat  <- factor(sample(c('a','b'), 500, TRUE))
y      <- 8*(treat == 'b') + rnorm(500, 100, 20)
f <- ols(y ~ treat*center)


lc <- levels(center)
contrast(f, list(treat='b', center=lc),
            list(treat='a', center=lc))


# Get 'Type III' contrast: average b - a treatment effect over
# centers, weighting centers equally (which is almost always
# an unreasonable thing to do)
contrast(f, list(treat='b', center=lc),
            list(treat='a', center=lc),
         type='average')


# Get 'Type II' contrast, weighting centers by the number of
# subjects per center.  Print the design contrast matrix used.
k <- contrast(f, list(treat='b', center=lc),
                 list(treat='a', center=lc),
              type='average', weights=table(center))
print(k, X=TRUE)
# Note: If other variables had interacted with either treat
# or center, we may want to list settings for these variables
```

```
# inside the list()'s, so as to not use default settings


# For a 4-treatment study, get all comparisons with treatment 'a'
treat  <- factor(sample(c('a','b','c','d'),  500, TRUE))
y      <- 8*(treat == 'b') + rnorm(500, 100, 20)
dd     <- datadist(treat, center); options(datadist='dd')
f <- ols(y ~ treat*center)
lt <- levels(treat)
contrast(f, list(treat=lt[-1]),
            list(treat=lt[ 1]),
         cnames=paste(lt[-1], lt[1], sep=':'), conf.int=1 - .05 / 3)


# Compare each treatment with average of all others
for(i in 1 : length(lt)) {
  cat('Comparing with', lt[i], '\n\n')
  print(contrast(f, list(treat=lt[-i]),
                    list(treat=lt[ i]), type='average'))
}
options(datadist=NULL)

# Six ways to get the same thing, for a variable that
# appears linearly in a model and does not interact with
# any other variables.  We estimate the change in y per
# unit change in a predictor x1.  Methods 4, 5 also
# provide confidence limits.  Method 6 computes nonparametric
# bootstrap confidence limits.  Methods 2-6 can work
# for models that are nonlinear or non-additive in x1.
# For that case more care is needed in choice of settings
# for x1 and the variables that interact with x1.


## Not run:
coef(fit)['x1']                            # method 1
diff(predict(fit, gendata(x1=c(0,1))))     # method 2
g <- Function(fit)                         # method 3
g(x1=1) - g(x1=0)
summary(fit, x1=c(0,1))                    # method 4
k <- contrast(fit, list(x1=1), list(x1=0)) # method 5
print(k, X=TRUE)
fit <- update(fit, x=TRUE, y=TRUE)         # method 6
b <- bootcov(fit, B=500)
contrast(fit, list(x1=1), list(x1=0))


# In a model containing age, race, and sex,
# compute an estimate of the mean response for a
# 50 year old male, averaged over the races using
# observed frequencies for the races as weights


f <- ols(y ~ age + race + sex)
```

```
contrast(f, list(age=50, sex='male', race=levels(race)),
         type='average', weights=table(race))

# For a Bayesian model get the highest posterior interval for the
# difference in two nonlinear functions of predicted values
# Start with the mean from a proportional odds model
g <- blrm(y ~ x)
M <- Mean(g)
contrast(g, list(x=1), list(x=0), fun=M)

# For the median we have to make sure that contrast can pass the
# per-posterior-draw vector of intercepts through
qu <- Quantile(g)
med <- function(lp, intercepts) qu(0.5, lp, intercepts=intercepts)
contrast(g, list(x=1), list(x=0), fun=med)

## End(Not run)


# Plot the treatment effect (drug - placebo) as a function of age
# and sex in a model in which age nonlinearly interacts with treatment
# for females only

set.seed(1)
n <- 800
treat <- factor(sample(c('drug','placebo'), n,TRUE))
sex   <- factor(sample(c('female','male'),  n,TRUE))
age   <- rnorm(n, 50, 10)
y     <- .05*age + (sex=='female')*(treat=='drug')*.05*abs(age-50) + rnorm(n)
f     <- ols(y ~ rcs(age,4)*treat*sex)
d     <- datadist(age, treat, sex); options(datadist='d')

# show separate estimates by treatment and sex

require(ggplot2)
ggplot(Predict(f, age, treat, sex='female'))
ggplot(Predict(f, age, treat, sex='male'))
ages  <- seq(35,65,by=5); sexes <- c('female','male')
w     <- contrast(f, list(treat='drug',    age=ages, sex=sexes),
                     list(treat='placebo', age=ages, sex=sexes))
# add conf.type="simultaneous" to adjust for having done 14 contrasts
xYplot(Cbind(Contrast, Lower, Upper) ~ age | sex, data=w,
       ylab='Drug - Placebo')
w <- as.data.frame(w[c('age','sex','Contrast','Lower','Upper')])
ggplot(w, aes(x=age, y=Contrast)) + geom_point() + facet_grid(sex ~ .) +
   geom_errorbar(aes(ymin=Lower, ymax=Upper), width=0)
ggplot(w, aes(x=age, y=Contrast)) + geom_line() + facet_grid(sex ~ .) +
   geom_ribbon(aes(ymin=Lower, ymax=Upper), width=0, alpha=0.15, linetype=0)
xYplot(Cbind(Contrast, Lower, Upper) ~ age, groups=sex, data=w,
       ylab='Drug - Placebo', method='alt bars')
options(datadist=NULL)
```

```
# Examples of type='joint' contrast tests

set.seed(1)
x1 <- rnorm(100)
x2 <- factor(sample(c('a','b','c'), 100, TRUE))
dd <- datadist(x1, x2); options(datadist='dd')
y  <- x1 + (x2=='b') + rnorm(100)

# First replicate a test statistic from anova()

f <- ols(y ~ x2)
anova(f)
contrast(f, list(x2=c('b','c')), list(x2='a'), type='joint')

# Repeat with a redundancy; compare a vs b, a vs c, b vs c

contrast(f, list(x2=c('a','a','b')), list(x2=c('b','c','c')), type='joint')

# Get a test of association of a continuous predictor with y
# First assume linearity, then cubic

f <- lrm(y>0 ~ x1 + x2)
anova(f)
contrast(f, list(x1=1), list(x1=0), type='joint')  # a minimum set of contrasts
xs <- seq(-2, 2, length=20)
contrast(f, list(x1=0), list(x1=xs), type='joint')

# All contrasts were redundant except for the first, because of
# linearity assumption

f <- lrm(y>0 ~ pol(x1,3) + x2, x=TRUE, y=TRUE)
anova(f)
anova(f, test='LR')   # discrepancy with Wald statistics points out a problem w/them

contrast(f, list(x1=0), list(x1=xs), type='joint')
print(contrast(f, list(x1=0), list(x1=xs), type='joint'), jointonly=TRUE)

# All contrasts were redundant except for the first 3, because of
# cubic regression assumption
# These Wald tests and intervals are not very accurate.  Although joint
# testing is not implemented in contrast(), individual profile likelihood
# confidence intervals and associted likelihood ratio tests are helpful:
# contrast(f, list(x1=0), list(x1=xs), conf.type='profile', plot_profile=TRUE)

# Now do something that is difficult to do without cryptic contrast
# matrix operations: Allow each of the three x2 groups to have a different
# shape for the x1 effect where x1 is quadratic.  Test whether there is
# a difference in mean levels of y for x2='b' vs. 'c' or whether
# the shape or slope of x1 is different between x2='b' and x2='c' regardless
# of how they differ when x2='a'.  In other words, test whether the mean
# response differs between group b and c at any value of x1.
# This is a 3 d.f. test (intercept, linear, quadratic effects) and is
# a better approach than subsetting the data to remove x2='a' then
```

```
# fitting a simpler model, as it uses a better estimate of sigma from
# all the data.

f <- ols(y ~ pol(x1,2) * x2)
anova(f)
contrast(f, list(x1=xs, x2='b'),
            list(x1=xs, x2='c'), type='joint')

# Note: If using a spline fit, there should be at least one value of
# x1 between any two knots and beyond the outer knots.
options(datadist=NULL)
```

---

cph *Cox Proportional Hazards Model and Extensions*

Description

Modification of Therneau's coxph function to fit the Cox model and its extension, the Andersen-Gill model. The latter allows for interval time-dependent covariables, time-dependent strata, and repeated events. The Survival method for an object created by cph returns an S function for computing estimates of the survival function. The Quantile method for cph returns an S function for computing quantiles of survival time (median, by default). The Mean method returns a function for computing the mean survival time. This function issues a warning if the last follow-up time is uncensored, unless a restricted mean is explicitly requested.

Usage

```
cph(formula = formula(data), data=environment(formula),
    weights, subset, na.action=na.delete,
    method=c("efron","breslow","exact","model.frame","model.matrix"),
    singular.ok=FALSE, robust=FALSE,
    model=FALSE, x=FALSE, y=FALSE, se.fit=FALSE,
    linear.predictors=TRUE, residuals=TRUE, nonames=FALSE,
    eps=1e-4, init, iter.max=10, tol=1e-9, surv=FALSE, time.inc,
    type=NULL, vartype=NULL, debug=FALSE, ...)

## S3 method for class 'cph'
Survival(object, ...)
# Evaluate result as g(times, lp, stratum=1, type=c("step","polygon"))

## S3 method for class 'cph'
Quantile(object, ...)
# Evaluate like h(q, lp, stratum=1, type=c("step","polygon"))

## S3 method for class 'cph'
Mean(object, method=c("exact","approximate"), type=c("step","polygon"),
          n=75, tmax, ...)
# E.g. m(lp, stratum=1, type=c("step","polygon"), tmax, \dots)
```

Arguments

`formula` an S formula object with a `Surv` object on the left-hand side. The `terms` can specify any S model formula with up to third-order interactions. The `strat` function may appear in the terms, as a main effect or an interacting factor. To stratify on both race and sex, you would include both terms `strat(race)` and `strat(sex)`. Stratification factors may interact with non-stratification factors; not all stratification terms need interact with the same modeled factors.

`object` an object created by `cph` with `surv=TRUE`

`data` name of an S data frame containing all needed variables. Omit this to use a data frame already in the S "search list".

`weights` case weights

`subset` an expression defining a subset of the observations to use in the fit. The default is to use all observations. Specify for example `age>50 & sex="male"` or `c(1:100,200:300)` respectively to use the observations satisfying a logical expression or those having row numbers in the given vector.

`na.action` specifies an S function to handle missing data. The default is the function `na.delete`, which causes observations with any variable missing to be deleted. The main difference between `na.delete` and the S-supplied function `na.omit` is that `na.delete` makes a list of the number of observations that are missing on each variable in the model. The `na.action` is usally specified by e.g. `options(na.action="na.delete")`.

`method` for `cph`, specifies a particular fitting method, `"model.frame"` instead to return the model frame of the predictor and response variables satisfying any subset or missing value checks, or `"model.matrix"` to return the expanded design matrix. The default is `"efron"`, to use Efron's likelihood for fitting the model.

For `Mean.cph`, `method` is `"exact"` to use numerical integration of the survival function at any linear predictor value to obtain a mean survival time. Specify `method="approximate"` to use an approximate method that is slower when `Mean.cph` is executing but then is essentially instant thereafter. For the approximate method, the area is computed for `n` points equally spaced between the min and max observed linear predictor values. This calculation is done separately for each stratum. Then the `n` pairs (X beta, area) are saved in the generated S function, and when this function is evaluated, the `approx` function is used to evaluate the mean for any given linear predictor values, using linear interpolation over the `n` X beta values.

`singular.ok` If `TRUE`, the program will automatically skip over columns of the X matrix that are linear combinations of earlier columns. In this case the coefficients for such columns will be NA, and the variance matrix will contain zeros. For ancillary calculations, such as the linear predictor, the missing coefficients are treated as zeros. The singularities will prevent many of the features of the `rms` library from working.

`robust` if `TRUE` a robust variance estimate is returned. Default is `TRUE` if the model includes a `cluster()` operative, `FALSE` otherwise.

`model` default is `FALSE`(false). Set to `TRUE` to return the model frame as element `model` of the fit object.

x — default is FALSE. Set to TRUE to return the expanded design matrix as element x (without intercept indicators) of the returned fit object.

y — default is FALSE. Set to TRUE to return the vector of response values (Surv object) as element y of the fit.

se.fit — default is FALSE. Set to TRUE to compute the estimated standard errors of the estimate of X beta and store them in element se.fit of the fit. The predictors are first centered to their means before computing the standard errors.

linear.predictors — set to FALSE to omit linear.predictors vector from fit

residuals — set to FALSE to omit residuals vector from fit

nonames — set to TRUE to not set names attribute for linear.predictors, residuals, se.fit, and rows of design matrix

eps — convergence criterion - change in log likelihood.

init — vector of initial parameter estimates. Defaults to all zeros. Special residuals can be obtained by setting some elements of init to MLEs and others to zero and specifying iter.max=1.

iter.max — maximum number of iterations to allow. Set to 0 to obtain certain null-model residuals.

tol — tolerance for declaring singularity for matrix inversion (available only when survival5 or later package is in effect)

surv — set to TRUE to compute underlying survival estimates for each stratum, and to store these along with standard errors of log Lambda(t), maxtime (maximum observed survival or censoring time), and surv.summary in the returned object. Set surv="summary" to only compute and store surv.summary, not survival estimates at each unique uncensored failure time. If you specify x=TRUE and y=TRUE, you can obtain predicted survival later, with accurate confidence intervals for any set of predictor values. The standard error information stored as a result of surv=TRUE are only accurate at the mean of all predictors. If the model has no covariables, these are of course OK. The main reason for using surv is to greatly speed up the computation of predicted survival probabilities as a function of the covariables, when accurate confidence intervals are not needed.

time.inc — time increment used in deriving surv.summary. Survival, number at risk, and standard error will be stored for t=0, time.inc, 2 time.inc, ..., maxtime, where maxtime is the maximum survival time over all strata. time.inc is also used in constructing the time axis in the survplot function (see below). The default value for time.inc is 30 if units(ftime) = "Day" or no units attribute has been attached to the survival time variable. If units(ftime) is a word other than "Day", the default for time.inc is 1 when it is omitted, unless maxtime<1, then maxtime/10 is used as time.inc. If time.inc is not given and maxtime/default time.inc > 25, time.inc is increased.

type — (for cph) applies if surv is TRUE or "summary". If type is omitted, the method consistent with method is used. See survfit.coxph (under survfit) or survfit.cph for details and for the definitions of values of type

For Survival, Quantile, Mean set to "polygon" to use linear interpolation instead of the usual step function. For Mean, the default of step will yield the sample mean in the case of no censoring and no covariables, if type="kaplan-meier"

was specified to cph. For method="exact", the value of type is passed to the generated function, and it can be overridden when that function is actually invoked. For method="approximate", Mean.cph generates the function different ways according to type, and this cannot be changed when the function is actually invoked.

vartype — see survfit.coxph

debug — set to TRUE to print debugging information related to model matrix construction. You can also use options(debug=TRUE).

... — other arguments passed to coxph.fit from cph. Ignored by other functions.

times — a scalar or vector of times at which to evaluate the survival estimates

lp — a scalar or vector of linear predictors (including the centering constant) at which to evaluate the survival estimates

stratum — a scalar stratum number or name (e.g., "sex=male") to use in getting survival probabilities

q — a scalar quantile or a vector of quantiles to compute

n — the number of points at which to evaluate the mean survival time, for method="approximate" in Mean.cph.

tmax — For Mean.cph, the default is to compute the overall mean (and produce a warning message if there is censoring at the end of follow-up). To compute a restricted mean life length, specify the truncation point as tmax. For method="exact", tmax is passed to the generated function and it may be overridden when that function is invoked. For method="approximate", tmax must be specified at the time that Mean.cph is run.

Details

If there is any strata by covariable interaction in the model such that the mean X beta varies greatly over strata, method="approximate" may not yield very accurate estimates of the mean in Mean.cph.

For method="approximate" if you ask for an estimate of the mean for a linear predictor value that was outside the range of linear predictors stored with the fit, the mean for that observation will be NA.

Value

For Survival, Quantile, or Mean, an S function is returned. Otherwise, in addition to what is listed below, formula/design information and the components maxtime, time.inc, units, model, x, y, se.fit are stored, the last 5 depending on the settings of options by the same names. The vectors or matrix stored if y=TRUE or x=TRUE have rows deleted according to subset and to missing data, and have names or row names that come from the data frame used as input data.

n — table with one row per stratum containing number of censored and uncensored observations

coef — vector of regression coefficients

stats	vector containing the named elements Obs, Events, Model L.R., d.f., P, Score, Score P, R2, Somers' Dxy, g-index, and gr, the g-index on the hazard ratio scale. R2 is the Nagelkerke R-squared, with division by the maximum attainable R-squared.
var	variance/covariance matrix of coefficients
linear.predictors	values of predicted X beta for observations used in fit, normalized to have overall mean zero, then having any offsets added
resid	martingale residuals
loglik	log likelihood at initial and final parameter values
score	value of score statistic at initial values of parameters
times	lists of times (if surv="T")
surv	lists of underlying survival probability estimates
std.err	lists of standard errors of estimate log-log survival
surv.summary	a 3 dimensional array if surv=TRUE. The first dimension is time ranging from 0 to maxtime by time.inc. The second dimension refers to strata. The third dimension contains the time-oriented matrix with Survival, n.risk (number of subjects at risk), and std.err (standard error of log-log survival).
center	centering constant, equal to overall mean of X beta.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
<fh@fharrell.com>

See Also

coxph, survival-internal, Surv, residuals.cph, cox.zph, survfit.cph, survest.cph, survfit.coxph, survplot, datadist, rms, rms.trans, anova.rms, summary.rms, Predict, fastbw, validate, calibrate, plot.Predict, ggplot.Predict, specs.rms, lrm, which.influence, na.delete, na.detail.response, print.cph, latex.cph, vif, ie.setup, GiniMd, dxy.cens, concordance

Examples

```
# Simulate data from a population model in which the log hazard
# function is linear in age and there is no age x sex interaction

require(survival)
require(ggplot2)
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('Male','Female'), n,
              rep=TRUE, prob=c(.6, .4)))
cens <- 15*runif(n)
```

```
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
dt <- -log(runif(n))/h
label(dt) <- 'Follow-up Time'
e <- ifelse(dt <= cens,1,0)
dt <- pmin(dt, cens)
units(dt) <- "Year"
dd <- datadist(age, sex)
options(datadist='dd')
S <- Surv(dt,e)

f <- cph(S ~ rcs(age,4) + sex, x=TRUE, y=TRUE)
cox.zph(f, "rank")             # tests of PH
anova(f)
ggplot(Predict(f, age, sex)) # plot age effect, 2 curves for 2 sexes
survplot(f, sex)             # time on x-axis, curves for x2
res <- resid(f, "scaledsch")
time <- as.numeric(dimnames(res)[[1]])
z <- loess(res[,4] ~ time, span=0.50)   # residuals for sex
plot(time, fitted(z))
lines(supsmu(time, res[,4]),lty=2)
plot(cox.zph(f,"identity"))    #Easier approach for last few lines
# latex(f)


f <- cph(S ~ age + strat(sex), surv=TRUE)
g <- Survival(f)   # g is a function
g(seq(.1,1,by=.1), stratum="sex=Male", type="poly") #could use stratum=2
med <- Quantile(f)
plot(Predict(f, age, fun=function(x) med(lp=x)))  #plot median survival

# Fit a model that is quadratic in age, interacting with sex as strata
# Compare standard errors of linear predictor values with those from
# coxph
# Use more stringent convergence criteria to match with coxph

f <- cph(S ~ pol(age,2)*strat(sex), x=TRUE, eps=1e-9, iter.max=20)
coef(f)
se <- predict(f, se.fit=TRUE)$se.fit
require(lattice)
xyplot(se ~ age | sex, main='From cph')
a <- c(30,50,70)
comb <- data.frame(age=rep(a, each=2),
                   sex=rep(levels(sex), 3))

p <- predict(f, comb, se.fit=TRUE)
comb$yhat  <- p$linear.predictors
comb$se    <- p$se.fit
z <- qnorm(.975)
comb$lower <- p$linear.predictors - z*p$se.fit
comb$upper <- p$linear.predictors + z*p$se.fit
comb

age2 <- age^2
```

```
f2 <- coxph(S ~ (age + age2)*strata(sex))
coef(f2)
se <- predict(f2, se.fit=TRUE)$se.fit
xyplot(se ~ age | sex, main='From coxph')
comb <- data.frame(age=rep(a, each=2), age2=rep(a, each=2)^2,
                   sex=rep(levels(sex), 3))
p <- predict(f2, newdata=comb, se.fit=TRUE)
comb$yhat <- p$fit
comb$se   <- p$se.fit
comb$lower <- p$fit - z*p$se.fit
comb$upper <- p$fit + z*p$se.fit
comb


# g <- cph(Surv(hospital.charges) ~ age, surv=TRUE)
# Cox model very useful for analyzing highly skewed data, censored or not
# m <- Mean(g)
# m(0)                           # Predicted mean charge for reference age


#Fit a time-dependent covariable representing the instantaneous effect
#of an intervening non-fatal event
rm(age)
set.seed(121)
dframe <- data.frame(failure.time=1:10, event=rep(0:1,5),
                     ie.time=c(NA,1.5,2.5,NA,3,4,NA,5,5,5),
                     age=sample(40:80,10,rep=TRUE))
z <- ie.setup(dframe$failure.time, dframe$event, dframe$ie.time)
S <- z$S
ie.status <- z$ie.status
attach(dframe[z$subs,])    # replicates all variables

f <- cph(S ~ age + ie.status, x=TRUE, y=TRUE)
#Must use x=TRUE,y=TRUE to get survival curves with time-dep. covariables


#Get estimated survival curve for a 50-year old who has an intervening
#non-fatal event at 5 days
new <- data.frame(S=Surv(c(0,5), c(5,999), c(FALSE,FALSE)), age=rep(50,2),
                  ie.status=c(0,1))
g <- survfit(f, new)
plot(c(0,g$time), c(1,g$surv[,2]), type='s',
     xlab='Days', ylab='Survival Prob.')
# Not certain about what columns represent in g$surv for survival5
# but appears to be for different ie.status
#or:
#g <- survest(f, new)
#plot(g$time, g$surv, type='s', xlab='Days', ylab='Survival Prob.')


#Compare with estimates when there is no intervening event
new2 <- data.frame(S=Surv(c(0,5), c(5, 999), c(FALSE,FALSE)), age=rep(50,2),
                   ie.status=c(0,0))
```

```
g2 <- survfit(f, new2)
lines(c(0,g2$time), c(1,g2$surv[,2]), type='s', lty=2)
#or:
#g2 <- survest(f, new2)
#lines(g2$time, g2$surv, type='s', lty=2)
detach("dframe[z$subs, ]")
options(datadist=NULL)
```

`cr.setup` *Continuation Ratio Ordinal Logistic Setup*

Description

Creates several new variables which help set up a dataset with an ordinal response variable y for use in fitting a forward continuation ratio (CR) model. The CR model can be fitted with binary logistic regression if each input observation is replicated the proper number of times according to the y value, a new binary y is computed that has at most one $y = 1$ per subject, and if a `cohort` variable is used to define the current qualifying condition for a cohort of subjects, e.g., $y \geq 2$. `cr.setup` creates the needed auxilliary variables. See `predab.resample` and `validate.lrm` for information about validating CR models (e.g., using the bootstrap to sample with replacement from the original subjects instead of the records used in the fit, validating the model separately for user-specified values of `cohort`).

Usage

```
cr.setup(y)
```

Arguments

`y`	a character, numeric, `category`, or `factor` vector containing values of the response variable. For `category` or `factor` variables, the `levels` of the variable are assumed to be listed in an ordinal way.

Value

a list with components `y, cohort, subs, reps`. `y` is a new binary variable that is to be used in the binary logistic fit. `cohort` is a `factor` vector specifying which cohort condition currently applies. `subs` is a vector of subscripts that can be used to replicate other variables the same way `y` was replicated. `reps` specifies how many times each original observation was replicated. `y, cohort, subs` are all the same length and are longer than the original `y` vector. `reps` is the same length as the original `y` vector. The `subs` vector is suitable for passing to `validate.lrm` or `calibrate`, which pass this vector under the name `cluster` on to `predab.resample` so that bootstrapping can be done by sampling with replacement from the original subjects rather than from the individual records created by `cr.setup`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

References

Berridge DM, Whitehead J: Analysis of failure time data with ordinal categories of response. Stat in Med 10:1703–1710, 1991.

See Also

lrm, glm, predab.resample

Examples

```
y <- c(NA, 10, 21, 32, 32)
cr.setup(y)


set.seed(171)
y <- sample(0:2, 100, rep=TRUE)
sex <- sample(c("f","m"),100,rep=TRUE)
sex <- factor(sex)
table(sex, y)
options(digits=5)
tapply(y==0, sex, mean)
tapply(y==1, sex, mean)
tapply(y==2, sex, mean)
cohort <- y>=1
tapply(y[cohort]==1, sex[cohort], mean)

u <- cr.setup(y)
Y <- u$y
cohort <- u$cohort
sex <- sex[u$subs]

lrm(Y ~ cohort + sex)

f <- lrm(Y ~ cohort*sex)   # saturated model - has to fit all data cells
f

#Prob(y=0|female):
# plogis(-.50078)
#Prob(y=0|male):
# plogis(-.50078+.11301)
#Prob(y=1|y>=1, female):
plogis(-.50078+.31845)
#Prob(y=1|y>=1, male):
plogis(-.50078+.31845+.11301-.07379)
```

```
combinations <- expand.grid(cohort=levels(cohort), sex=levels(sex))
combinations
p <- predict(f, combinations, type="fitted")
p
p0 <- p[c(1,3)]
p1 <- p[c(2,4)]
p1.unconditional <- (1 - p0) *p1
p1.unconditional
p2.unconditional <- 1 - p0 - p1.unconditional
p2.unconditional


## Not run:
dd <- datadist(inputdata)   # do this on non-replicated data
options(datadist='dd')
pain.severity <- inputdata$pain.severity
u <- cr.setup(pain.severity)
# inputdata frame has age, sex with pain.severity
attach(inputdata[u$subs,])  # replicate age, sex
# If age, sex already available, could do age <- age[u$subs] etc., or
# age <- rep(age, u$reps), etc.
y      <- u$y
cohort <- u$cohort
dd     <- datadist(dd, cohort)       # add to dd
f <- lrm(y ~ cohort + age*sex)       # ordinary cont. ratio model
g <- lrm(y ~ cohort*sex + age, x=TRUE,y=TRUE) # allow unequal slopes for
                                     # sex across cutoffs
cal <- calibrate(g, cluster=u$subs, subset=cohort=='all')
# subs makes bootstrap sample the correct units, subset causes
# Predicted Prob(pain.severity=0) to be checked for calibration

## End(Not run)
```

`datadist` *Distribution Summaries for Predictor Variables*

Description

For a given set of variables or a data frame, determines summaries of variables for effect and plotting ranges, values to adjust to, and overall ranges for `Predict`, `plot.Predict`, `ggplot.Predict`, `summary.rms`, `survplot`, and `nomogram.rms`. If `datadist` is called before a model fit and the resulting object pointed to with `options(datadist="name")`, the data characteristics will be stored with the fit by `Design()`, so that later predictions and summaries of the fit will not need to access the original data used in the fit. Alternatively, you can specify the values for each variable in the model when using these 3 functions, or specify the values of some of them and let the functions look up the remainder (of say adjustmemt levels) from an object created by `datadist`. The best method is probably to run `datadist` once before any models are fitted, storing the distribution summaries for all potential variables. Adjustment values are `0` for binary variables, the most frequent category (or optionally the first category level) for categorical (`factor`) variables, the middle level for

ordered factor variables, and medians for continuous variables. See descriptions of q.display and q.effect for how display and effect ranges are chosen for continuous variables.

Usage

```
datadist(..., data, q.display, q.effect=c(0.25, 0.75),
         adjto.cat=c('mode','first'), n.unique=10)

## S3 method for class 'datadist'
print(x, ...)
# options(datadist="dd")
# used by summary, plot, survplot, sometimes predict
# For dd substitute the name of the result of datadist
```

Arguments

...	a list of variable names, separated by commas, a single data frame, or a fit with Design information. The first element in this list may also be an object created by an earlier call to datadist; then the later variables are added to this datadist object. For a fit object, the variables named in the fit are retrieved from the active data frame or from the location pointed to by data=frame number or data="data frame name". For print, is ignored.
data	a data frame or a search position. If data is a search position, it is assumed that a data frame is attached in that position, and all its variables are used. If you specify both individual variables in ... and data, the two sets of variables are combined. Unless the first argument is a fit object, data must be an integer.
q.display	set of two quantiles for computing the range of continuous variables to use in displaying regression relationships. Defaults are q and $1-q$, where $q = 10/max(n, 200)$, and n is the number of non-missing observations. Thus for $n < 200$, the .05 and .95 quantiles are used. For $n \geq 200$, the 10th smallest and 10th largest values are used. If you specify q.display, those quantiles are used whether or not $n < 200$.
q.effect	set of two quantiles for computing the range of continuous variables to use in estimating regression effects. Defaults are c(.25,.75), which yields inter-quartile-range odds ratios, etc.
adjto.cat	default is "mode", indicating that the modal (most frequent) category for categorical (factor) variables is the adjust-to setting. Specify "first" to use the first level of factor variables as the adjustment values. In the case of many levels having the maximum frequency, the first such level is used for "mode".
n.unique	variables having n.unique or fewer unique values are considered to be discrete variables in that their unique values are stored in the values list. This will affect how functions such as nomogram.Design determine whether variables are discrete or not.
x	result of datadist

Details

For categorical variables, the 7 limits are set to character strings (factors) which correspond to `c(NA,adjto.level,NA,1,k,1,k)`, where `k` is the number of levels. For ordered variables with numeric levels, the limits are set to `c(L,M,H,L,H,L,H)`, where `L` is the lowest level, `M` is the middle level, and `H` is the highest level.

Value

a list of class `"datadist"` with the following components

`limits` a $7 \times k$ vector, where k is the number of variables. The 7 rows correspond to the low value for estimating the effect of the variable, the value to adjust the variable to when examining other variables, the high value for effect, low value for displaying the variable, the high value for displaying it, and the overall lowest and highest values.

`values` a named list, with one vector of unique values for each numeric variable having no more than `n.unique` unique values

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

`rms`, `rms.trans`, `describe`, `Predict`, `summary.rms`

Examples

```
## Not run:
d <- datadist(data=1)         # use all variables in search pos. 1
d <- datadist(x1, x2, x3)
page(d)                       # if your options(pager) leaves up a pop-up
                              # window, this is a useful guide in analyses
d <- datadist(data=2)         # all variables in search pos. 2
d <- datadist(data=my.data.frame)
d <- datadist(my.data.frame)  # same as previous.  Run for all potential vars.
d <- datadist(x2, x3, data=my.data.frame)   # combine variables
d <- datadist(x2, x3, q.effect=c(.1,.9), q.display=c(0,1))
# uses inter-decile range odds ratios,
# total range of variables for regression function plots
d <- datadist(d, z)           # add a new variable to an existing datadist
options(datadist="d")         #often a good idea, to store info with fit
f <- ols(y ~ x1*x2*x3)


options(datadist=NULL)        #default at start of session
f <- ols(y ~ x1*x2)
```

```
d <- datadist(f)              #info not stored in `f'
d$limits["Adjust to","x1"] <- .5   #reset adjustment level to .5
options(datadist="d")


f <- lrm(y ~ x1*x2, data=mydata)
d <- datadist(f, data=mydata)
options(datadist="d")


f <- lrm(y ~ x1*x2)           #datadist not used - specify all values for
summary(f, x1=c(200,500,800), x2=c(1,3,5))         # obtaining predictions
plot(Predict(f, x1=200:800, x2=3))  # or ggplot()


# Change reference value to get a relative odds plot for a logistic model
d$limits$age[2] <- 30    # make 30 the reference value for age
# Could also do: d$limits["Adjust to","age"] <- 30
fit <- update(fit)   # make new reference value take effect
plot(Predict(fit, age, ref.zero=TRUE, fun=exp),
     ylab='Age=x:Age=30 Odds Ratio')   # or ggplot()

## End(Not run)
```

ExProb *Function Generators For Exceedance and Survival Probabilities*

Description

For an `orm` object `ExProb` generates a function for computing the estimates of the function Prob(Y>=y) given one or more values of the linear predictor using the reference (median) intercept. This function can optionally be evaluated at only a set of user-specified `y` values, otherwise a right-step function is returned. There is a plot method for plotting the step functions, and if more than one linear predictor was evaluated multiple step functions are drawn. `ExProb` is especially useful for `nomogram`. `Survival` generates a similar function but for computing survival probabilities Prob(Y>y) and adding an origin of zero. Plotting of survival curves is done with a `survplot` method. For survival estimation when interval censoring is present, times are taken as interval midpoints with intervals corresponding to intercepts in the model.

Optionally a normal approximation (normality for the linear predictor) for a confidence interval for exceedance probabilities will be computed, if `conf.int > 0` is specified to the function generated from calling `ExProb` or `Survival`. For `ExProb`, a `"lims"` attribute is included in the result computed by the derived cumulative probability function. For `Survival`, the result is a data frame if `conf.int` is specified or both time and the requested linear predictor are varying. In the data frame the limits are variables `lower` and `upper`.

Usage

```
ExProb(object, ...)
```

```
## S3 method for class 'orm'
ExProb(object, codes = FALSE, ...)

## S3 method for class 'ExProb'
plot(x, ..., data=NULL,
                      xlim=NULL, xlab=x$yname, ylab=expression(Prob(Y>=y)),
                      col=par('col'), col.vert='gray85', pch=20,
                      pch.data=21, lwd=par('lwd'), lwd.data=lwd,
                      lty.data=2, key=TRUE)

## S3 method for class 'orm'
Survival(object, ...)
```

Arguments

`object`	a fit object from `orm`. For `Survival` the fit may be from `orm.fit`. This is used to estimate survival curves when there are no predictors in the model. In the case the link function (`family` argument to `orm.fit`) does not affect survival probabilities but does affect confidence limits. To get the same confidence intervals as `survival:survfit.formula` use `ormfit(y=, family='loglog')` to correspond to `survfit(..., conf.type='log-log')`.
`codes`	if `TRUE`, `ExProb` use the integer codes $1, 2, \ldots, k$ for the k-level response instead of its original unique values
`...`	ignored for `ExProb`. Passed to `plot` for `plot.ExProb`
`data`	Specify `data` if you want to add stratified empirical probabilities to the graph. If `data` is a numeric vector, it is assumed that no groups are present. Otherwise `data` must be a list or data frame where the first variable is the grouping variable (corresponding to what made the linear predictor vary) and the second variable is the data vector for the `y` variable. The rows of data should be sorted to be in order of the linear predictor argument.
`x`	an object created by running the function created by `ExProb`
`xlim`	limits for x-axis; default is range of observed `y`
`xlab`	x-axis label
`ylab`	y-axis label
`col`	color for horizontal lines and points
`col.vert`	color for vertical discontinuities
`pch`	plotting symbol for predicted curves
`lwd`	line width for predicted curves
`pch.data`, `lwd.data`, `lty.data`	plotting parameters for data
`key`	set to `FALSE` to suppress key in plot if `data` is given

Value

`ExProb` and `Survival` return an R function. Running the function returns an object of class "`ExProb`" for `ExProb`, or a data frame or vector for `Survival`.

Author(s)

Frank Harrell and Shengxin Tu

See Also

orm, Quantile.orm

Examples

```
set.seed(1)
x1 <- runif(200)
yvar <- x1 + runif(200)
f <- orm(yvar ~ x1)
d <- ExProb(f)
lp <- predict(f, newdata=data.frame(x1=c(.2,.8)))
w <- d(lp)
s1 <- abs(x1 - .2) < .1
s2 <- abs(x1 - .8) < .1
plot(w, data=data.frame(x1=c(rep(.2, sum(s1)), rep(.8, sum(s2))),
                        yvar=c(yvar[s1], yvar[s2])))

qu <- Quantile(f)
abline(h=c(.1,.5), col='gray80')
abline(v=qu(.5, lp), col='gray80')
abline(v=qu(.9, lp), col='green')
## Not run:
  Y <- Ocens(dtime, ifelse(censored, Inf, dtime))
  f <- orm(Y ~ x, family='loglog')
  s <- Survival(f)
  s()  # all times
  s(times=c(1, 3))
  d <- data.frame(x=2:4)
  s(X=predict(f, d, conf.int=0.95)  # all times
  s(lp=predict(f, d))  # same surv estimates, no CLs
  # use s(..., forcedf=TRUE) to force output to be a data.frame

## End(Not run)
```

---

fastbw *Fast Backward Variable Selection*

Description

Performs a slightly inefficient but numerically stable version of fast backward elimination on factors, using a method based on Lawless and Singhal (1978). This method uses the fitted complete model and computes approximate Wald statistics by computing conditional (restricted) maximum likelihood estimates assuming multivariate normality of estimates. fastbw deletes factors, not columns of the design matrix. Factors requiring multiple d.f. will be retained or dropped as a group. The function prints the deletion statistics for each variable in turn, and prints approximate

parameter estimates for the model after deleting variables. The approximation is better when the number of factors deleted is not large. For `ols`, the approximation is exact for regression coefficients, and standard errors are only off by a factor equal to the ratio of the mean squared error estimate for the reduced model to the original mean squared error estimate for the full model.

If the fit was from `ols`, `fastbw` will compute the usual R^2 statistic for each model.

Usage

```
fastbw(fit, rule=c("aic", "p"),
       type=c("residual", "individual", "total"),
       sls=.05, aics=0, eps=.Machine$double.eps,
       k.aic=2, force=NULL)

## S3 method for class 'fastbw'
print(x, digits=4, estimates=TRUE, ...)
```

Arguments

`fit`	fit object with `Varcov(fit)` defined (e.g., from `ols`, `lrm`, `cph`, `psm`, `glmD`)
`rule`	Stopping rule. Defaults to `"aic"` for Akaike's information criterion. Use `rule="p"` to use P-values
`type`	Type of statistic on which to base the stopping rule. Default is `"residual"` for the pooled residual chi-square. Use `type="individual"` to use Wald chi-square of individual factors.
`sls`	Significance level for staying in a model if `rule="p"`. Default is .05.
`aics`	For `rule="aic"`, variables are deleted until the chi-square - `k.aic` times d.f. would rise above `aics`. Default `aics` is zero to use the ordinary AIC. Set `aics` to say 10000 to see all variables deleted in order of descending importance.
`eps`	Singularity criterion, default is `1E-14`.
`k.aic`	multiplier to compute AIC, default is 2. To use BIC, set `k.aic` equal to $\log(n)$, where n is the effective sample size (number of events for survival models).
`force`	a vector of integers specifying parameters forced to be in the model, not counting intercept(s)
`x`	result of `fastbw`
`digits`	number of significant digits to print
`estimates`	set to `FALSE` to suppress printing table of approximate coefficients, SEs, etc., after variable deletions
`...`	ignored

Value

a list with an attribute `kept` if `bw=TRUE`, and the following components:

`result`	matrix of statistics with rows in order of deletion.
`names.kept`	names of factors kept in final model.

factors.kept	the subscripts of factors kept in the final model
factors.deleted	opposite of factors.kept.
parms.kept	column numbers in design matrix corresponding to parameters kept in the final model.
parms.deleted	opposite of parms.kept.
coefficients	vector of approximate coefficients of reduced model.
var	approximate covariance matrix for reduced model.
Coefficients	matrix of coefficients of all models. Rows correspond to the successive models examined and columns correspond to the coefficients in the full model. For variables not in a particular sub-model (row), the coefficients are zero.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Lawless, J. F. and Singhal, K. (1978): Efficient screening of nonnormal regression models. Biometrics 34:318–327.

See Also

rms, ols, lrm, cph, psm, validate, solvet, rmsMisc

Examples

```
## Not run:
fastbw(fit, optional.arguments)     # print results
z <- fastbw(fit, optional.args)     # typically used in simulations
lm.fit(X[,z$parms.kept], Y)         # least squares fit of reduced model

## End(Not run)
```

Function *Compose an S Function to Compute X beta from a Fit*

Description

Function is a class of functions for creating other S functions. Function.rms is the method for creating S functions to compute X beta, based on a model fitted with rms in effect. Like latexrms, Function.rms simplifies restricted cubic spline functions and factors out terms in second-order interactions. Function.rms will not work for models that have third-order interactions involving restricted cubic splines. Function.cph is a particular method for handling fits from cph, for which

an intercept (the negative of the centering constant) is added to the model. sascode is a function that takes an S function such as one created by Function and does most of the editing to turn the function definition into a fragment of SAS code for computing X beta from the fitted model, along with assignment statements that initialize predictors to reference values. perlcode similarly creates Perl code to evaluate a fitted regression model.

Usage

```
## S3 method for class 'rms'
Function(object, intercept=NULL,
 digits=max(8, .Options$digits), posterior.summary=c('mean', 'median', 'mode'), ...)
## S3 method for class 'cph'
Function(object, intercept=-object$center, ...)

# Use result as fun(predictor1=value1, predictor2=value2, \dots)

sascode(object, file='', append=FALSE)

perlcode(object)
```

Arguments

object	a fit created with rms in effect
intercept	an intercept value to use (not allowed to be specified to Function.cph). The intercept is usually retrieved from the regression coefficients automatically.
digits	number of significant digits to use for coefficients and knot locations
posterior.summary	if using a Bayesian model fit such as from blrm, specifies whether to use posterior mode/mean/median parameter estimates in generating the function
file	name of a file in which to write the SAS code. Default is to write to standard output.
append	set to TRUE to have sascode append code to an existing file named file.
...	arguments to pass to Function.rms from Function.cph

Value

Function returns an S-Plus function that can be invoked in any usual context. The function has one argument per predictor variable, and the default values of the predictors are set to adjust-to values (see datadist). Multiple predicted X beta values may be calculated by specifying vectors as arguments to the created function. All non-scalar argument values must have the same length. perlcode returns a character string with embedded newline characters.

Author(s)

Frank Harrell, Jeremy Stephens, and Thomas Dupont
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

latexrms, transcan, predict.rms, rms, rms.trans

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1331)
x1 <- exp(rnorm(100))
x2 <- factor(sample(c('a','b'),100,rep=TRUE))
dd <- datadist(x1, x2)
options(datadist='dd')
y  <- log(x1)^2+log(x1)*(x2=='b')+rnorm(100)/4
f  <- ols(y ~ pol(log(x1),2)*x2)
f$coef
g  <- Function(f, digits=5)
g
sascode(g)
cat(perlcode(g), '\n')
g()
g(x1=c(2,3), x2='b')   #could omit x2 since b is default category
predict(f, expand.grid(x1=c(2,3),x2='b'))
g8 <- Function(f)   # default is 8 sig. digits
g8(x1=c(2,3), x2='b')
options(datadist=NULL)


## Not run:
require(survival)
# Make self-contained functions for computing survival probabilities
# using a log-normal regression
f <- psm(Surv(d.time, death) ~ rcs(age,4)*sex, dist='gaussian')
g <- Function(f)
surv <- Survival(f)
# Compute 2 and 5-year survival estimates for 50 year old male
surv(c(2,5), g(age=50, sex='male'))

## End(Not run)
```

---

gendata *Generate Data Frame with Predictor Combinations*

Description

If nobs is not specified, allows user to specify predictor settings by e.g. age=50, sex="male", and any omitted predictors are set to reference values (default=median for continuous variables, first level for categorical ones - see datadist). If any predictor has more than one value given, expand.grid is called to generate all possible combinations of values, unless expand=FALSE. If nobs is given, a data frame is first generated which has nobs of adjust-to values duplicated. Then an editor window is opened which allows the user to subset the variable names down to ones which

she intends to vary (this streamlines the `data.ed` step). Then, if any predictors kept are discrete and `viewvals=TRUE`, a window (using `page`) is opened defining the possible values of this subset, to facilitate data editing. Then the `data.ed` function is invoked to allow interactive overriding of predictor settings in the `nobs` rows. The subset of variables are combined with the other predictors which were not displayed with `data.ed`, and a final full data frame is returned. `gendata` is most useful for creating a `newdata` data frame to pass to `predict`.

Usage

```
gendata(fit, ..., nobs, viewvals=FALSE, expand=TRUE, factors)
```

Arguments

`fit`	a fit object created with `rms` in effect
`...`	predictor settings, if `nobs` is not given.
`nobs`	number of observations to create if doing it interactively using X-windows
`viewvals`	if `nobs` is given, set `viewvals=TRUE` to open a window displaying the possible value of categorical predictors
`expand`	set to `FALSE` to prevent `expand.grid` from being called, and to instead just convert to a data frame.
`factors`	a list containing predictor settings with their names. This is an alternative to specifying the variables separately in `...`. Unlike the usage of `...`, variables getting default ranges in `factors` should have `NA` as their value.

Details

if you have a variable in `...` that is named `n`, `no`, `nob`, `nob`, add `nobs=FALSE` to the invocation to prevent that variable from being misrecognized as `nobs`

Value

a data frame with all predictors, and an attribute `names.subset` if `nobs` is specified. This attribute contains the vector of variable names for predictors which were passed to `de` and hence were allowed to vary. If neither `nobs` nor any predictor settings were given, returns a data frame with adjust-to values.

Side Effects

optionally writes to the terminal, opens X-windows, and generates a temporary file using `sink`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

predict.rms, survest.cph, survest.psm, rmsMisc, expand.grid, de, page, print.datadist, Predict

Examples

```
set.seed(1)
age <- rnorm(200, 50, 10)
sex <- factor(sample(c('female','male'),200,TRUE))
race <- factor(sample(c('a','b','c','d'),200,TRUE))
y <- sample(0:1, 200, TRUE)
dd <- datadist(age,sex,race)
options(datadist="dd")
f <- lrm(y ~ age*sex + race)
gendata(f)
gendata(f, age=50)
d <- gendata(f, age=50, sex="female")  # leave race=reference category
d <- gendata(f, age=c(50,60), race=c("b","a"))  # 4 obs.
d$Predicted <- predict(f, d, type="fitted")
d      # Predicted column prints at the far right
options(datadist=NULL)
## Not run:
d <- gendata(f, nobs=5, view=TRUE)        # 5 interactively defined obs.
d[,attr(d,"names.subset")]             # print variables which varied
predict(f, d)

## End(Not run)
```

---

ggplot.npsurv *Title Plot npsurv Nonparametric Survival Curves Using ggplot2*

Description

Title Plot npsurv Nonparametric Survival Curves Using ggplot2

Usage

```
## S3 method for class 'npsurv'
ggplot(
  data,
  mapping,
  conf = c("bands", "none"),
  trans = c("identity", "logit", "probit", "loglog"),
  logt = FALSE,
  curtail = c(0, 1),
  xlab,
  ylab = "Survival Probability",
  abbrev.label = FALSE,
```

```
  levels.only = TRUE,
  alpha = 0.15,
  facet = FALSE,
  npretty = 10,
  onlydata = FALSE,
  ...,
  environment
)
```

Arguments

data	the result of npsurv
mapping	unused
conf	set to "none" to suppress confidence bands
trans	the name of a transformation for the survival probabilities to use in drawing the y-axis scale. The default is no transformation, and other choices are "logit", "probit", "loglog". "loglog" represents $-log(-log(S(t)))$
logt	set to TRUE to use a log scale for the x-axis
curtail	set to a (lower, upper) 2-vector to curtail survival probabilities and confidence limits before transforming and plotting
xlab	x-axis label, the default coming from fit
ylab	y-axis label, the default coming from fit
abbrev.label	set to TRUE to abbreviate strata levels
levels.only	set to FALSE to keep the original strata name in the levels
alpha	transparency for confidence bands
facet	when strata are present, set to TRUE to facet them rather than using colors on one panel
npretty	the number of major tick mark labels to be constructed by scales::breaks_pretty() or pretty(). For transformed scales, twice this number is used.
onlydata	set to TRUE to return the data frame to be plotted, and no plot
...	ignored
environment	unused

Value

a ggplot2 object, if onlydata=FALSE

Author(s)

Frank Harrell

Examples

```
set.seed(1)
g <- c(rep('a', 500), rep('b', 500))
y <- exp(-1 + 2 * (g == 'b') + rlogis(1000) / 3)
f <- npsurv(Surv(y) ~ g)
ggplot(f, trans='logit', logt=TRUE)
```

---

ggplot.Predict *Plot Effects of Variables Estimated by a Regression Model Fit Using ggplot2*

Description

Uses ggplot2 graphics to plot the effect of one or two predictors on the linear predictor or X beta scale, or on some transformation of that scale. The first argument specifies the result of the Predict function. The predictor is always plotted in its original coding.

If rdata is given, a spike histogram is drawn showing the location/density of data values for the x-axis variable. If there is a groups (superposition) variable that generated separate curves, the data density specific to each class of points is shown. This assumes that the second variable was a factor variable. The histograms are drawn by histSpikeg.

To plot effects instead of estimates (e.g., treatment differences as a function of interacting factors) see contrast.rms and summary.rms.

Usage

```
## S3 method for class 'Predict'
ggplot(data, mapping, formula=NULL, groups=NULL,
     aestype=c('color', 'linetype'),
     conf=c('fill', 'lines'),
     conflinetype=1,
     varypred=FALSE, sepdiscrete=c('no', 'list', 'vertical', 'horizontal'),
     subset, xlim., ylim., xlab, ylab,
     colorscale=function(...) scale_color_manual(...,
       values=c("#000000", "#E69F00", "#56B4E9",
                "#009E73","#F0E442", "#0072B2", "#D55E00", "#CC79A7")),
     colfill='black',
     rdata=NULL, anova=NULL, pval=FALSE, size.anova=4,
     adj.subtitle, size.adj=2.5, perim=NULL, nlevels=3,
     flipxdiscrete=TRUE,
     legend.position='right', legend.label=NULL,
     vnames=c('labels','names'), abbrev=FALSE, minlength=6,
     layout=NULL, addlayer,
     histSpike.opts=list(frac=function(f) 0.01 +
         0.02 * sqrt(f - 1)/sqrt(max(f, 2) - 1), side=1, nint=100),
     type=NULL, ggexpr=FALSE, height=NULL, width=NULL, ..., environment)
```

Arguments

data	a data frame created by Predict
mapping	kept because of ggplot generic setup. If specified it will be assumed to be formula.
formula	a ggplot faceting formula of the form vertical variables ~ horizontal variables, with variables separated by * if there is more than one variable on a side. If omitted, the formula will be built using assumptions on the list of variables that varied in the Predict call. When plotting multiple panels (for separate predictors), formula may be specified but by default no formula is constructed.
groups	an optional character string containing the name of one of the variables in data that is to be used as a grouping (superpositioning) variable. Set groups=FALSE to suppress superpositioning. By default, the second varying variable is used for superpositioning groups. You can also specify a length 2 string vector of variable names specifying two dimensions of superpositioning, identified by different aesthetics corresponding to the aestype argument. When plotting effects of more than one predictor, groups is a character string that specifies a single variable name in data that can be used to form panels. Only applies if using rbind to combine several Predict results. If there is more than one groups variable, confidence bands are suppressed because ggplot2:geom_ribbon does not handle the aesthetics correctly.
aestype	a string vector of aesthetic names corresponding to variables in the groups vector. Default is to use, in order, color, and linetype. Other permissible values are size, shape.
conf	specify conf="line" to show confidence bands with lines instead of filled ribbons, the default
conflinetype	specify an alternative linetype for confidence intervals if conf="line"
varypred	set to TRUE if data is the result of passing multiple Predict results, that represent different predictors, to rbind.Predict. This will cause the .set. variable created by rbind to be copied to the .predictor. variable.
sepdiscrete	set to something other than "no" to create separate graphics for continuous and discrete predictors. For discrete predictors, horizontal dot charts are produced. This allows use of the ggplot2 facet_wrap function to make better use of space. If sepdiscrete="list", a list of two grid graphics objects is returned if both types of predictors are present (otherwise one object for the type that existed in the model). Set sepdiscrete="vertical" to put the two types of plots into one graphical object with continuous predictors on top and given a fraction of space relative to the number of continuous vs. number of discrete variables. Set sepdiscrete="horizontal" to get a horizontal arrangements with continuous variables on the left.
subset	a subsetting expression for restricting the rows of data that are used in plotting. For example, predictions may have been requested for males and females but one wants to plot only females.
xlim.	This parameter is seldom used, as limits are usually controlled with Predict. Usually given as its legal abbreviation xlim. One reason to use xlim is to plot a factor variable on the x-axis that was created with the cut2 function with

the levels.mean option, with val.lev=TRUE specified to plot.Predict. In this case you may want the axis to have the range of the original variable values given to cut2 rather than the range of the means within quantile groups.

ylim. Range for plotting on response variable axis. Computed by default. Usually specified using its legal definition ylim.

xlab Label for x-axis. Default is one given to asis, rcs, etc., which may have been the "label" attribute of the variable.

ylab Label for y-axis. If fun is not given, default is "log Odds" for lrm, "log Relative Hazard" for cph, name of the response variable for ols, TRUE or log(TRUE) for psm, or "X * Beta" otherwise. Specify ylab=NULL to omit y-axis labels.

colorscale a ggplot2 discrete scale function, e.g. function(...) scale_color_brewer(..., palette='Set1', type='qual'). The default is the colorblind-friendly palette including black in http://www.cookbook-r.com/Graphs/Colors_(ggplot2). If you get an error "insufficient values in manual scale", which occurs when there are more than 8 groups, just specify colorscale=function(...){} to use default colors.

colfill a single character string or number specifying the fill color to use for geom_ribbon for shaded confidence bands. Alpha transparency of 0.2 is applied to any color specified.

rdata a data frame containing the original raw data on which the regression model were based, or at least containing the x-axis and grouping variable. If rdata is present and contains the needed variables, the original data are added to the graph in the form of a spike histogram using histSpikeg in the Hmisc package.

anova an object returned by anova.rms. If anova is specified, the overall test of association for predictor plotted is added as text to each panel, located at the spot at which the panel is most empty unless there is significant empty space at the top or bottom of the panel; these areas are given preference.

pval specify pval=TRUE for anova to include not only the test statistic but also the P-value

size.anova character size for the test statistic printed on the panel, mm

adj.subtitle Set to FALSE to suppress subtitling the graph with the list of settings of non-graphed adjustment values. Subtitles appear as captions with ggplot2 using labs(caption=).

size.adj Size of adjustment settings in subtitles in mm. Default is 2.5.

perim perim specifies a function having two arguments. The first is the vector of values of the first variable that is about to be plotted on the x-axis. The second argument is the single value of the variable representing different curves, for the current curve being plotted. The function's returned value must be a logical vector whose length is the same as that of the first argument, with values TRUE if the corresponding point should be plotted for the current curve, FALSE otherwise. See one of the latter examples. perim only applies if predictors were specified to Predict.

nlevels	when groups and formula are not specified, if any panel variable has nlevels or fewer values, that variable is converted to a groups (superpositioning) variable. Set nlevels=0 to prevent this behavior. For other situations, a non-numeric x-axis variable with nlevels or fewer unique values will cause a horizontal dot plot to be drawn instead of an x-y plot unless flipxdiscrete=FALSE.
flipxdiscrete	see nlevels
legend.position	"right" (the default for single-panel plots), "left", "bottom", "top", a two-element numeric vector, or "none" to suppress. For multi-panel plots the default is "top", and a legend only appears for the first (top left) panel.
legend.label	if omitted, group variable labels will be used for label the legend. Specify legend.label=FALSE to suppress using a legend name, or a character string or expression to specify the label. Can be a vector is there is more than one grouping variable.
vnames	applies to the case where multiple plots are produced separately by predictor. Set to 'names' to use variable names instead of labels for these small plots.
abbrev	set to true to abbreviate levels of predictors that are categorical to a minimum length of minlength
minlength	see abbrev
layout	for multi-panel plots a 2-vector specifying the number of rows and number of columns. If omitted will be computed from the number of panels to make as square as possible.
addlayer	a ggplot2 expression consisting of one or more layers to add to the current plot
histSpike.opts	a list containing named elements that specifies parameters to histSpikeg when rdata is given. The col parameter is usually derived from other plotting information and not specified by the user.
type	a value ("l","p","b") to override default choices related to showing or connecting points. Especially useful for discrete x coordinate variables.
ggexpr	set to TRUE to have the function return the character string(s) constructed to invoke ggplot without executing the commands
height, width	used if plotly is in effect, to specify the plotly image in pixels. Default is to let plotly size the image.
...	ignored
environment	ignored; used to satisfy rules because of the generic ggplot

Value

an object of class "ggplot2" ready for printing. For the case where predictors were not specified to Predict, sepdiscrete=TRUE, and there were both continuous and discrete predictors in the model, a list of two graphics objects is returned.

Note

If plotting the effects of all predictors you can reorder the panels using for example p <- Predict(fit); p$.predictor. <-factor(p$.predictor., v) where v is a vector of predictor names specified in the desired order.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Fox J, Hong J (2009): Effect displays in R for multinomial and proportional-odds logit models: Extensions to the effects package. J Stat Software 32 No. 1.

See Also

Predict, rbind.Predict, datadist, predictrms, anova.rms, contrast.rms, summary.rms, rms, rmsMisc, plot.Predict, labcurve, histSpikeg, ggplot, Overview

Examples

```
require(ggplot2)
n <- 350     # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
label(age)            <- 'Age'      # label is in Hmisc
label(cholesterol)    <- 'Total Cholesterol'
label(blood.pressure) <- 'Systolic Blood Pressure'
label(sex)            <- 'Sex'
units(cholesterol)    <- 'mg/dl'   # uses units.default in Hmisc
units(blood.pressure) <- 'mmHg'

# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
  (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male')) +
  .01 * (blood.pressure - 120)
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)

ddist <- datadist(age, blood.pressure, cholesterol, sex)
options(datadist='ddist')

fit <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)),
               x=TRUE, y=TRUE)
an <- anova(fit)
# Plot effects in two vertical sub-panels with continuous predictors on top
# ggplot(Predict(fit), sepdiscrete='vertical')
# Plot effects of all 4 predictors with test statistics from anova, and P
ggplot(Predict(fit), anova=an, pval=TRUE)
# ggplot(Predict(fit), rdata=llist(blood.pressure, age))
# spike histogram plot for two of the predictors
```

```
# p <- Predict(fit, name=c('age','cholesterol'))   # Make 2 plots
# ggplot(p)

# p <- Predict(fit, age=seq(20,80,length=100), sex, conf.int=FALSE)
#                        # Plot relationship between age and log
                         # odds, separate curve for each sex,
# ggplot(p, subset=sex=='female' | age > 30)
# No confidence interval, suppress estimates for males <= 30

# p <- Predict(fit, age, sex)
# ggplot(p, rdata=llist(age,sex))
                         # rdata= allows rug plots (1-dimensional scatterplots)
                         # on each sex's curve, with sex-
                         # specific density of age
                         # If data were in data frame could have used that
# p <- Predict(fit, age=seq(20,80,length=100), sex='male', fun=plogis)
                         # works if datadist not used
# ggplot(p, ylab=expression(hat(P)))
                         # plot predicted probability in place of log odds
# per <- function(x, y) x >= 30
# ggplot(p, perim=per)       # suppress output for age < 30 but leave scale alone

# Do ggplot2 faceting a few different ways
p <- Predict(fit, age, sex, blood.pressure=c(120,140,160),
             cholesterol=c(180,200,215))
# ggplot(p)
ggplot(p, cholesterol ~ blood.pressure)
# ggplot(p, ~ cholesterol + blood.pressure)
# color for sex, line type for blood.pressure:
ggplot(p, groups=c('sex', 'blood.pressure'))
# Add legend.position='top' to allow wider plot
# Map blood.pressure to line thickness instead of line type:
# ggplot(p, groups=c('sex', 'blood.pressure'), aestype=c('color', 'size'))

# Plot the age effect as an odds ratio
# comparing the age shown on the x-axis to age=30 years

# ddist$limits$age[2] <- 30    # make 30 the reference value for age
# Could also do: ddist$limits["Adjust to","age"] <- 30
# fit <- update(fit)   # make new reference value take effect
# p <- Predict(fit, age, ref.zero=TRUE, fun=exp)
# ggplot(p, ylab='Age=x:Age=30 Odds Ratio',
#        addlayer=geom_hline(yintercept=1, col=gray(.8)) +
#                 geom_vline(xintercept=30, col=gray(.8)) +
#                 scale_y_continuous(trans='log',
#                       breaks=c(.5, 1, 2, 4, 8))))

# Compute predictions for three predictors, with superpositioning or
# conditioning on sex, combined into one graph

p1 <- Predict(fit, age, sex)
p2 <- Predict(fit, cholesterol, sex)
p3 <- Predict(fit, blood.pressure, sex)
```

```
p <- rbind(age=p1, cholesterol=p2, blood.pressure=p3)
ggplot(p, groups='sex', varypred=TRUE, adj.subtitle=FALSE)
# ggplot(p, groups='sex', varypred=TRUE, adj.subtitle=FALSE, sepdiscrete='vert')

## Not run:
# For males at the median blood pressure and cholesterol, plot 3 types
# of confidence intervals for the probability on one plot, for varying age
ages <- seq(20, 80, length=100)
p1 <- Predict(fit, age=ages, sex='male', fun=plogis)  # standard pointwise
p2 <- Predict(fit, age=ages, sex='male', fun=plogis,
              conf.type='simultaneous')               # simultaneous
p3 <- Predict(fit, age=c(60,65,70), sex='male', fun=plogis,
              conf.type='simultaneous')               # simultaneous 3 pts
# The previous only adjusts for a multiplicity of 3 points instead of 100
f <- update(fit, x=TRUE, y=TRUE)
g <- bootcov(f, B=500, coef.reps=TRUE)
p4 <- Predict(g, age=ages, sex='male', fun=plogis)    # bootstrap percentile
p <- rbind(Pointwise=p1, 'Simultaneous 100 ages'=p2,
           'Simultaneous     3 ages'=p3, 'Bootstrap nonparametric'=p4)
# as.data.frame so will call built-in ggplot
ggplot(as.data.frame(p), aes(x=age, y=yhat)) + geom_line() +
 geom_ribbon(data=p, aes(ymin=lower, ymax=upper), alpha=0.2, linetype=0)+
 facet_wrap(~ .set., ncol=2)

# Plots for a parametric survival model
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('Male','Female'), n,
              rep=TRUE, prob=c(.6, .4)))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
t <- -log(runif(n))/h
label(t) <- 'Follow-up Time'
e <- ifelse(t<=cens,1,0)
t <- pmin(t, cens)
units(t) <- "Year"
ddist <- datadist(age, sex)
require(survival)
Srv <- Surv(t,e)

# Fit log-normal survival model and plot median survival time vs. age
f <- psm(Srv ~ rcs(age), dist='lognormal')
med <- Quantile(f)       # Creates function to compute quantiles
                         # (median by default)
p <- Predict(f, age, fun=function(x) med(lp=x))
ggplot(p, ylab="Median Survival Time")
# Note: confidence intervals from this method are approximate since
# they don't take into account estimation of scale parameter


# Fit an ols model to log(y) and plot the relationship between x1
```

```
# and the predicted mean(y) on the original scale without assuming
# normality of residuals; use the smearing estimator
# See help file for rbind.Predict for a method of showing two
# types of confidence intervals simultaneously.
# Add raw data scatterplot to graph
set.seed(1)
x1 <- runif(300)
x2 <- runif(300)
ddist <- datadist(x1, x2); options(datadist='ddist')
y  <- exp(x1 + x2 - 1 + rnorm(300))
f <- ols(log(y) ~ pol(x1,2) + x2)
r <- resid(f)
smean <- function(yhat)smearingEst(yhat, exp, res, statistic='mean')
formals(smean) <- list(yhat=numeric(0), res=r[! is.na(r)])
#smean$res <- r[! is.na(r)]   # define default res argument to function
ggplot(Predict(f, x1, fun=smean), ylab='Predicted Mean on y-scale',
   addlayer=geom_point(aes(x=x1, y=y), data.frame(x1, y)))
# Had ggplot not added a subtitle (i.e., if x2 were not present), you
# could have done ggplot(Predict(), ylab=...) + geom_point(...)

## End(Not run)

# Make an 'interaction plot', forcing the x-axis variable to be
# plotted at integer values but labeled with category levels
n <- 100
set.seed(1)
gender <- c(rep('male', n), rep('female',n))
m <- sample(c('a','b'), 2*n, TRUE)
d <-  datadist(gender, m); options(datadist='d')
anxiety <- runif(2*n) + .2*(gender=='female') + .4*(gender=='female' & m=='b')
tapply(anxiety, llist(gender,m), mean)
f <- ols(anxiety ~ gender*m)
p <- Predict(f, gender, m)
# ggplot(p)     # horizontal dot chart; usually preferred for categorical predictors
# ggplot(p, flipxdiscrete=FALSE)  # back to vertical
ggplot(p, groups='gender')
ggplot(p, ~ m, groups=FALSE, flipxdiscrete=FALSE)

options(datadist=NULL)

## Not run:
# Example in which separate curves are shown for 4 income values
# For each curve the estimated percentage of voters voting for
# the democratic party is plotted against the percent of voters
# who graduated from college.  Data are county-level percents.

incomes <- seq(22900, 32800, length=4)
# equally spaced to outer quintiles
p <- Predict(f, college, income=incomes, conf.int=FALSE)
ggplot(p, xlim=c(0,35), ylim=c(30,55))

# Erase end portions of each curve where there are fewer than 10 counties having
# percent of college graduates to the left of the x-coordinate being plotted,
```

```
# for the subset of counties having median family income with 1650
# of the target income for the curve

show.pts <- function(college.pts, income.pt) {
  s <- abs(income - income.pt) < 1650  #assumes income known to top frame
  x <- college[s]
  x <- sort(x[!is.na(x)])
  n <- length(x)
  low <- x[10]; high <- x[n-9]
  college.pts >= low & college.pts <= high
}

ggplot(p, xlim=c(0,35), ylim=c(30,55), perim=show.pts)

# Rename variables for better plotting of a long list of predictors
f <- ...
p <- Predict(f)
re <- c(trt='treatment', diabet='diabetes', sbp='systolic blood pressure')

for(n in names(re)) {
  names(p)[names(p)==n] <- re[n]
  p$.predictor.[p$.predictor.==n] <- re[n]
  }
ggplot(p)

## End(Not run)
```

---

gIndex *Calculate Total and Partial g-indexes for an rms Fit*

Description

gIndex computes the total g-index for a model based on the vector of linear predictors, and the partial g-index for each predictor in a model. The latter is computed by summing all the terms involving each variable, weighted by their regression coefficients, then computing Gini's mean difference on this sum. For example, a regression model having age and sex and age*sex on the right hand side, with corresponding regression coefficients b_1, b_2, b_3 will have the g-index for age computed from Gini's mean difference on the product of age $\times(b_1 + b_3 w)$ where w is an indicator set to one for observations with sex not equal to the reference value. When there are nonlinear terms associated with a predictor, these terms will also be combined.

A print method is defined, and there is a plot method for displaying g-indexes using a dot chart.

These functions use Hmisc::GiniMd.

Usage

```
gIndex(object, partials=TRUE, type=c('ccterms', 'cterms', 'terms'),
           lplabel=if(length(object$scale) && is.character(object$scale))
           object$scale[1] else 'X*Beta',
```

```
            fun, funlabel=if(missing(fun)) character(0) else
            deparse(substitute(fun)),
            postfun=if(length(object$scale)==2) exp else NULL,
            postlabel=if(length(postfun))
            ifelse(missing(postfun),
                   if((length(object$scale) > 1) &&
                      is.character(object$scale)) object$scale[2] else
                      'Anti-log',
                      deparse(substitute(postfun))) else character(0),
            ...)

## S3 method for class 'gIndex'
print(x, digits=4, abbrev=FALSE,
 vnames=c("names","labels"), ...)

## S3 method for class 'gIndex'
plot(x, what=c('pre', 'post'),
 xlab=NULL, pch=16, rm.totals=FALSE,
sort=c('descending', 'ascending', 'none'), ...)
```

Arguments

object	result of an rms fitting function
partials	set to FALSE to suppress computation of partial gs
type	defaults to 'ccterms' which causes partial discrimination indexes to be computed after maximally combining all related main effects and interactions. The is usually the only way that makes sense when considering partial linear predictors. Specify type='cterms' to only combine a main effect with interactions containing it, not also with other main effects connected through interactions. Use type='terms' to separate interactions into their own effects.
lplabel	a replacement for default values such as "X*Beta" or "log odds"/
fun	an optional function to transform the linear predictors before computing the total (only) g. When this is present, a new component gtrans is added to the attributes of the object resulting from gIndex.
funlabel	a character string label for fun, otherwise taken from the function name itself
postfun	a function to transform g such as exp (anti-log), which is the default for certain models such as the logistic and Cox models
postlabel	a label for postfun
...	For gIndex, passed to predict.rms. Ignored for print. Passed to dotchart2 for plot.
x	an object created by gIndex (for print or plot)
digits	causes rounding to the digits decimal place
abbrev	set to TRUE to abbreviate labels if vname="labels"
vnames	set to "labels" to print predictor labels instead of names
what	set to "post" to plot the transformed g-index if there is one (e.g., ratio scale)

xlab	x-axis label; constructed by default
pch	plotting character for point
rm.totals	set to TRUE to remove the total g-index when plotting
sort	specifies how to sort predictors by g-index; default is in descending order going down the dot chart

Details

For stratification factors in a Cox proportional hazards model, there is no contribution of variation towards computing a partial g except from terms that interact with the stratification variable.

Value

gIndex returns a matrix of class "gIndex" with auxiliary information stored as attributes, such as variable labels. GiniMd returns a scalar.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
<fh@fharrell.com>

References

David HA (1968): Gini's mean difference rediscovered. Biometrika 55:573–575.

See Also

predict.rms,GiniMd

Examples

```
set.seed(1)
n <- 40
x <- 1:n
w <- factor(sample(c('a','b'), n, TRUE))
u <- factor(sample(c('A','B'), n, TRUE))
y <- .01*x + .2*(w=='b') + .3*(u=='B') + .2*(w=='b' & u=='B') + rnorm(n)/5
dd <- datadist(x,w,u); options(datadist='dd')
f <- ols(y ~ x*w*u, x=TRUE, y=TRUE)
f
anova(f)
z <- list()
for(type in c('terms','cterms','ccterms'))
  {
    zc <- predict(f, type=type)
    cat('type:', type, '\n')
    print(zc)
    z[[type]] <- zc
```

```
  }

zc <- z$cterms
GiniMd(zc[, 1])
GiniMd(zc[, 2])
GiniMd(zc[, 3])
GiniMd(f$linear.predictors)
g <- gIndex(f)
g
g['Total',]
gIndex(f, partials=FALSE)
gIndex(f, type='cterms')
gIndex(f, type='terms')

y <- y > .8
f <- lrm(y ~ x * w * u, x=TRUE, y=TRUE, reltol=1e-5)

gIndex(f, fun=plogis, funlabel='Prob[y=1]')

# Manual calculation of combined main effect + interaction effort of
# sex in a 2x2 design with treatments A B, sexes F M,
# model -.1 + .3*(treat=='B') + .5*(sex=='M') + .4*(treat=='B' & sex=='M')

set.seed(1)
X <- expand.grid(treat=c('A','B'), sex=c('F', 'M'))
a <- 3; b <- 7; c <- 13; d <- 5
X <- rbind(X[rep(1, a),], X[rep(2, b),], X[rep(3, c),], X[rep(4, d),])
y <- with(X, -.1 + .3*(treat=='B') + .5*(sex=='M') + .4*(treat=='B' & sex=='M'))
f <- ols(y ~ treat*sex, data=X, x=TRUE)
gIndex(f, type='cterms')
k <- coef(f)
b1 <- k[2]; b2 <- k[3]; b3 <- k[4]
n <- nrow(X)
( (a+b)*c*abs(b2) + (a+b)*d*abs(b2+b3) + c*d*abs(b3))/(n*(n-1)/2 )

# Manual calculation for combined age effect in a model with sex,
# age, and age*sex interaction

a <- 13; b <- 7
sex <- c(rep('female',a), rep('male',b))
agef <- round(runif(a, 20, 30))
agem <- round(runif(b, 20, 40))
age  <- c(agef, agem)
y <- (sex=='male') + age/10 - (sex=='male')*age/20
f <- ols(y ~ sex*age, x=TRUE)
f
gIndex(f, type='cterms')
k <- coef(f)
b1 <- k[2]; b2 <- k[3]; b3 <- k[4]
n <- a + b
sp <- function(w, z=w) sum(outer(w, z, function(u, v) abs(u-v)))

( abs(b2)*sp(agef) + abs(b2+b3)*sp(agem) + 2*sp(b2*agef, (b2+b3)*agem) ) / (n*(n-1))
```

```
( abs(b2)*GiniMd(agef)*a*(a-1) + abs(b2+b3)*GiniMd(agem)*b*(b-1) +
  2*sp(b2*agef, (b2+b3)*agem) ) / (n*(n-1))

## Not run:
# Compare partial and total g-indexes over many random fits
plot(NA, NA, xlim=c(0,3), ylim=c(0,3), xlab='Global',
     ylab='x1 (black)  x2 (red)  x3 (green)  x4 (blue)')
abline(a=0, b=1, col=gray(.9))
big <- integer(3)
n <- 50   # try with n=7 - see lots of exceptions esp. for interacting var
for(i in 1:100) {
   x1 <- runif(n)
   x2 <- runif(n)
   x3 <- runif(n)
   x4 <- runif(n)
   y  <- x1 + x2 + x3 + x4 + 2*runif(n)
   f <- ols(y ~ x1*x2+x3+x4, x=TRUE)
   # f <- ols(y ~ x1+x2+x3+x4, x=TRUE)   # also try this
   w <- gIndex(f)[,1]
   gt <- w['Total']
   points(gt, w['x1, x2'])
   points(gt, w['x3'], col='green')
   points(gt, w['x4'], col='blue')
   big[1] <- big[1] + (w['x1, x2'] > gt)
   big[2] <- big[2] + (w['x3'] > gt)
   big[3] <- big[3] + (w['x4'] > gt)
   }
print(big)

## End(Not run)

options(datadist=NULL)
```

`Glm` *rms Version of glm*

Description

This function saves `rms` attributes with the fit object so that `anova.rms`, `Predict`, etc. can be used just as with `ols` and other fits. No `validate` or `calibrate` methods exist for `Glm` though.

Usage

```
Glm(
  formula,
  family = gaussian,
  data = environment(formula),
  weights,
  subset,
```

```
  na.action = na.delete,
  start = NULL,
  offset = NULL,
  control = glm.control(...),
  model = TRUE,
  method = "glm.fit",
  x = FALSE,
  y = TRUE,
  contrasts = NULL,
  ...
)
```

Arguments

formula, family, data, weights, subset, na.action, start, offset, control, model, method, x, y, contrasts
: see stats::glm(); for print x is the result of Glm

...
: ignored

Details

For the print method, format of output is controlled by the user previously running options(prType="lang") where lang is "plain" (the default), "latex", or "html".

Value

a fit object like that produced by stats::glm() but with rms attributes and a class of "rms", "Glm", "glm", and "lm". The g element of the fit object is the g-index.

See Also

stats::glm(),Hmisc::GiniMd(), prModFit(), stats::residuals.glm

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
f <- glm(counts ~ outcome + treatment, family=poisson())
f
anova(f)
summary(f)
f <- Glm(counts ~ outcome + treatment, family=poisson())
# could have had rcs( ) etc. if there were continuous predictors
f
anova(f)
summary(f, outcome=c('1','2','3'), treatment=c('1','2','3'))
```

Gls *Fit Linear Model Using Generalized Least Squares*

Description

This function fits a linear model using generalized least squares. The errors are allowed to be correlated and/or have unequal variances. `Gls` is a slightly enhanced version of the Pinheiro and Bates `gls` function in the `nlme` package to make it easy to use with the rms package and to implement cluster bootstrapping (primarily for nonparametric estimates of the variance-covariance matrix of the parameter estimates and for nonparametric confidence limits of correlation parameters).

For the `print` method, format of output is controlled by the user previously running `options(prType="lang")` where `lang` is `"plain"` (the default), `"latex"`, or `"html"`. When using html with Quarto or RMarkdown, `results='asis'` need not be written in the chunk header.

Usage

```
Gls(model, data, correlation, weights, subset, method, na.action=na.omit,
    control, verbose, B=0, dupCluster=FALSE, pr=FALSE, x=FALSE)

## S3 method for class 'Gls'
print(x, digits=4, coefs=TRUE, title, ...)
```

Arguments

`model`	a two-sided linear formula object describing the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right.
`data`	an optional data frame containing the variables named in `model`, `correlation`, `weights`, and `subset`. By default the variables are taken from the environment from which `gls` is called.
`correlation`	an optional `corStruct` object describing the within-group correlation structure. See the documentation of `corClasses` for a description of the available `corStruct` classes. If a grouping variable is to be used, it must be specified in the `form` argument to the `corStruct` constructor. Defaults to `NULL`, corresponding to uncorrelated errors.
`weights`	an optional `varFunc` object or one-sided formula describing the within-group heteroscedasticity structure. If given as a formula, it is used as the argument to `varFixed`, corresponding to fixed variance weights. See the documentation on `varClasses` for a description of the available `varFunc` classes. Defaults to `NULL`, corresponding to homoscesdatic errors.
`subset`	an optional expression indicating which subset of the rows of `data` should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
`method`	a character string. If `"REML"` the model is fit by maximizing the restricted log-likelihood. If `"ML"` the log-likelihood is maximized. Defaults to `"REML"`.

na.action	a function that indicates what should happen when the data contain NAs. The default action (na.omit) results in deletion of observations having any of the variables of interest missing.
control	a list of control values for the estimation algorithm to replace the default values returned by the function glsControl. Defaults to an empty list.
verbose	an optional logical value. If TRUE information on the evolution of the iterative algorithm is printed. Default is FALSE.
B	number of bootstrap resamples to fit and store, default is none
dupCluster	set to TRUE to have Gls when bootstrapping to consider multiply-sampled clusters as if they were one large cluster when fitting using the gls algorithm
pr	set to TRUE to show progress of bootstrap resampling
x	for Gls set to TRUE to store the design matrix in the fit object; otherwise the result of Gls
digits	number of significant digits to print
coefs	specify coefs=FALSE to suppress printing the table of model coefficients, standard errors, etc. Specify coefs=n to print only the first n regression coefficients in the model.
title	a character string title to be passed to prModFit
...	ignored

Details

The na.delete function will not work with Gls due to some nuance in the model.frame.default function. This probably relates to na.delete storing extra information in the "na.action" attribute of the returned data frame.

Value

an object of classes Gls, rms, and gls representing the linear model fit. Generic functions such as print, plot, ggplot, and summary have methods to show the results of the fit. See glsObject for the components of the fit. The functions resid, coef, and fitted can be used to extract some of its components. Gls returns the following components not returned by gls: Design, assign, formula (see arguments), B (see arguments), bootCoef (matrix of B bootstrapped coefficients), boot.Corr (vector of bootstrapped correlation parameters), Nboot (vector of total sample size used in each bootstrap (may vary if have unbalanced clusters), and var (sample variance-covariance matrix of bootstrapped coefficients). The g-index is also stored in the returned object under the name "g".

Author(s)

Jose Pinheiro, Douglas Bates <bates@stat.wisc.edu>, Saikat DebRoy, Deepayan Sarkar, R-core <R-core@R-project.org>, Frank Harrell <fh@fharrell.com>, Patrick Aboyoun

References

Pinheiro J, Bates D (2000): Mixed effects models in S and S-Plus. New York: Springer-Verlag.

See Also

gls glsControl, glsObject, varFunc, corClasses, varClasses, GiniMd, prModFit, logLik.Gls

Examples

```
## Not run:
require(ggplot2)
ns  <- 20  # no. subjects
nt  <- 10  # no. time points/subject
B   <- 10  # no. bootstrap resamples
           # usually do 100 for variances, 1000 for nonparametric CLs
rho <- .5  # AR(1) correlation parameter
V <- matrix(0, nrow=nt, ncol=nt)
V <- rho^abs(row(V)-col(V))   # per-subject correlation/covariance matrix

d <- expand.grid(tim=1:nt, id=1:ns)
d$trt <- factor(ifelse(d$id <= ns/2, 'a', 'b'))
true.beta <- c(Intercept=0,tim=.1,'tim^2'=0,'trt=b'=1)
d$ey  <- true.beta['Intercept'] + true.beta['tim']*d$tim +
  true.beta['tim^2']*(d$tim^2) +  true.beta['trt=b']*(d$trt=='b')
set.seed(13)
library(MASS)   # needed for mvrnorm
d$y <- d$ey + as.vector(t(mvrnorm(n=ns, mu=rep(0,nt), Sigma=V)))

dd <- datadist(d); options(datadist='dd')
f <- Gls(y ~ pol(tim,2) + trt, correlation=corCAR1(form= ~tim | id),
         data=d, B=B)
f
AIC(f)
f$var      # bootstrap variances
f$varBeta  # original variances
summary(f)
anova(f)
ggplot(Predict(f, tim, trt))
# v <- Variogram(f, form=~tim|id, data=d)
nlme:::summary.gls(f)$tTable   # print matrix of estimates etc.

options(datadist=NULL)

## End(Not run)
```

---

groupkm *Kaplan-Meier Estimates vs. a Continuous Variable*

Description

Function to divide x (e.g. age, or predicted survival at time u created by survest) into g quantile groups, get Kaplan-Meier estimates at time u (a scaler), and to return a matrix with columns x=mean x in quantile, n=number of subjects, events=no. events, and KM=K-M survival at time u, std.err

= s.e. of -log K-M. Confidence intervals are based on -log S(t). Instead of supplying g, the user can supply the minimum number of subjects to have in the quantile group (m, default=50). If cuts is given (e.g. cuts=c(0,.1,.2,...,.9,.1)), it overrides m and g. Calls Therneau's survfitKM in the survival package to get Kaplan-Meiers estimates and standard errors.

Usage

```
groupkm(x, Srv, m=50, g, cuts, u,
        pl=FALSE, loglog=FALSE, conf.int=.95, xlab, ylab,
        lty=1, add=FALSE, cex.subtitle=.7, ...)
```

Arguments

x	variable to stratify
Srv	a Surv object - n x 2 matrix containing survival time and event/censoring 1/0 indicator. Units of measurement come from the "units" attribute of the survival time variable. "Day" is the default.
m	desired minimum number of observations in a group
g	number of quantile groups
cuts	actual cuts in x, e.g. c(0,1,2) to use [0,1), [1,2].
u	time for which to estimate survival
pl	TRUE to plot results
loglog	set to TRUE to plot log(-log(survival)) instead of survival
conf.int	defaults to .95 for 0.95 confidence bars. Set to FALSE to suppress bars.
xlab	if pl=TRUE, is x-axis label. Default is label(x) or name of calling argument
ylab	if pl=TRUE, is y-axis label. Default is constructed from u and time units attribute.
lty	line time for primary line connecting estimates
add	set to TRUE if adding to an existing plot
cex.subtitle	character size for subtitle. Default is .7. Use FALSE to suppress subtitle.
...	plotting parameters to pass to the plot and errbar functions

Value

matrix with columns named x (mean predictor value in interval), n (sample size in interval), events (number of events in interval), KM (Kaplan-Meier estimate), std.err (standard error of -log KM)

See Also

survfit, errbar, cut2, Surv, units

Examples

```
require(survival)
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50))
d.time <- -log(runif(n))/h
label(d.time) <- 'Follow-up Time'
e <- ifelse(d.time <= cens,1,0)
d.time <- pmin(d.time, cens)
units(d.time) <- "Year"
groupkm(age, Surv(d.time, e), g=10, u=5, pl=TRUE)
#Plot 5-year K-M survival estimates and 0.95 confidence bars by
#decile of age.  If omit g=10, will have >= 50 obs./group.
```

---

hazard.ratio.plot *Hazard Ratio Plot*

Description

The `hazard.ratio.plot` function repeatedly estimates Cox regression coefficients and confidence limits within time intervals. The log hazard ratios are plotted against the mean failure/censoring time within the interval. Unless `times` is specified, the number of time intervals will be $\max(round(d/e), 2)$, where d is the total number of events in the sample. Efron's likelihood is used for estimating Cox regression coefficients (using `coxph.fit`). In the case of tied failure times, some intervals may have a point in common.

Usage

```
hazard.ratio.plot(x, Srv, which, times=, e=30, subset,
                  conf.int=.95, legendloc=NULL, smooth=TRUE, pr=FALSE, pl=TRUE,
                  add=FALSE, ylim, cex=.5, xlab="t", ylab, antilog=FALSE, ...)
```

Arguments

`x`	a vector or matrix of predictors
`Srv`	a `Surv` object
`which`	a vector of column numbers of `x` for which to estimate hazard ratios across time and make plots. The default is to do so for all predictors. Whenever one predictor is displayed, all other predictors in the `x` matrix are adjusted for (with a separate adjustment form for each time interval).
`times`	optional vector of time interval endpoints. Example: `times=c(1,2,3)` uses intervals `[0,1)`, `[1,2)`, `[2,3)`, `[3+)`. If times is omitted, uses intervals containing `e` events
`e`	number of events per time interval if times not given

subset	vector used for subsetting the entire analysis, e.g. subset=sex=="female"
conf.int	confidence interval coverage
legendloc	location for legend. Omit to use mouse, "none" for none, "ll" for lower left of graph, or actual x and y coordinates (e.g. c(2,3))
smooth	also plot the super–smoothed version of the log hazard ratios
pr	defaults to FALSE to suppress printing of individual Cox fits
pl	defaults to TRUE to plot results
add	add this plot to an already existing plot
ylim	vector of y-axis limits. Default is computed to include confidence bands.
cex	character size for legend information, default is 0.5
xlab	label for x-axis, default is "t"
ylab	label for y-axis, default is "Log Hazard Ratio" or "Hazard Ratio", depending on antilog.
antilog	default is FALSE. Set to TRUE to plot anti-log, i.e., hazard ratio.
...	optional graphical parameters

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

cox.zph, residuals.cph, survival-internal, cph, coxph, Surv

Examples

```
require(survival)
n <- 500
set.seed(1)
age <- 50 + 12*rnorm(n)
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50))
d.time <- -log(runif(n))/h
label(d.time) <- 'Follow-up Time'
e <- ifelse(d.time <= cens,1,0)
d.time <- pmin(d.time, cens)
units(d.time) <- "Year"
hazard.ratio.plot(age, Surv(d.time,e), e=20, legendloc='ll')
```

`ie.setup` *Intervening Event Setup*

Description

Creates several new variables which help set up a dataset for modeling with `cph` or `coxph` when there is a single binary time-dependent covariable which turns on at a given time, and stays on. This is typical when analyzing the impact of an intervening event. `ie.setup` creates a `Surv` object using the start time, stop time format. It also creates a binary indicator for the intervening event, and a variable called `subs` that is useful when `attach`-ing a dataframe. `subs` has observation numbers duplicated for subjects having an intervening event, so those subject's baseline covariables (that are not time-dependent) can be duplicated correctly.

Usage

```
ie.setup(failure.time, event, ie.time, break.ties=FALSE)
```

Arguments

`failure.time`	a numeric variable containing the event or censoring times for the terminating event
`event`	a binary (0/1) variable specifying whether observations had the terminating event (event=1) or were censored (event=0)
`ie.time`	intervening event times. For subjects having no intervening events, the corresponding values of ie.time must be NA.
`break.ties`	Occasionally intervening events are recorded as happening at exactly the same time as the termination of follow-up for some subjects. The `Surv` and `Surv` functions will not allow this. To randomly break the ties by subtracting a random number from such tied intervening event times, specify `break.ties=TRUE`. The random number is uniform between zero and the minimum difference between any two untied `failure.times`.

Value

a list with components `S, ie.status, subs, reps`. `S` is a `Surv` object containing start and stop times for intervals of observation, along with event indicators. `ie.status` is one if the intervening event has occurred at the start of the interval, zero otherwise. `subs` is a vector of subscripts that can be used to replicate other variables the same way `S` was replicated. `reps` specifies how many times each original observation was replicated. `S, ie.status, subs` are all the same length (at least the number of rows for `S` is) and are longer than the original `failure.time` vector. `reps` is the same length as the original `failure.time` vector. The `subs` vector is suitable for passing to `validate.lrm` or `calibrate`, which pass this vector under the name `cluster` on to `predab.resample` so that bootstrapping can be done by sampling with replacement from the original subjects rather than from the individual records created by `ie.setup`.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

cph, coxph, Surv, cr.setup, predab.resample

Examples

```
failure.time <- c(1 ,   2,   3)
event        <- c(1 ,   1,   0)
ie.time      <- c(NA, 1.5, 2.5)

z <- ie.setup(failure.time, event, ie.time)
S <- z$S
S
ie.status <- z$ie.status
ie.status
z$subs
z$reps
## Not run:
attach(input.data.frame[z$subs,])   #replicates all variables
f <- cph(S ~ age + sex + ie.status)
# Instead of duplicating rows of data frame, could do this:
attach(input.data.frame)
z <- ie.setup(failure.time, event, ie.time)
s <- z$subs
age <- age[s]
sex <- sex[s]
f <- cph(S ~ age + sex + ie.status)

## End(Not run)
```

---

impactPO *Impact of Proportional Odds Assumpton*

Description

Checks the impact of the proportional odds assumption by comparing predicted cell probabilities from a PO model with those from a multinomial or partial proportional odds logistic model that relax assumptions. For a given model formula, fits the model with both `lrm` and either `nnet::multinom` or `VGAM::vglm` or both, and obtains predicted cell probabilities for the PO and relaxed models on the `newdata` data frame. A `print` method formats the output.

Usage

```
impactPO(
  formula,
  relax = if (missing(nonpo)) "multinomial" else "both",
  nonpo,
  newdata,
  data = environment(formula),
  minfreq = 15,
  B = 0,
  ...
)
```

Arguments

formula	a model formula. To work properly with multinom or vglm the terms should have completely specified knot locations if a spline function is being used.
relax	defaults to "both" if nonpo is given, resulting in fitting two relaxed models. Set relax to "multinomial" or "ppo" to fit only one relaxed model. The multinomial model does not assume PO for any predictor.
nonpo	a formula with no left hand side variable, specifying the variable or variables for which PO is not assumed. Specifying nonpo results in a relaxed fit that is a partial PO model fitted with VGAM::vglm.
newdata	a data frame or data table with one row per covariate setting for which predictions are to be made
data	data frame containing variables to fit; default is the frame in which formula is found
minfreq	minimum sample size to allow for the least frequent category of the dependent variable. If the observed minimum frequency is less than this, the Hmisc::combine.levels() function will be called to combine enough consecutive levels so that this minimum frequency is achieved.
B	number of bootstrap resamples to do to get confidence intervals for differences in predicted probabilities for relaxed methods vs. PO model fits. Default is not to run the bootstrap. When running the bootstrap make sure that all model variables are explicitly in data= so that selection of random subsets of data will call along the correct rows for all predictors.
...	other parameters to pass to lrm and multinom

Details

Since partial proportional odds models and especially multinomial logistic models can have many parameters, it is not feasible to use this model comparison approach when the number of levels of the dependent variable Y is large. By default, the function will use Hmisc::combine.levels() to combine consecutive levels if the lowest frequency category of Y has fewer than minfreq observations.

Value

an impactPO object which is a list with elements estimates, stats, mad, newdata, nboot, and boot. estimates is a data frame containing the variables and values in newdata in a tall and thin format with additional variable method ("PO", "Multinomial", "PPO"), y (current level of the dependent variable), and Probability (predicted cell probability for covariate values and value of y in the current row). stats is a data frame containing Deviance the model deviance, d.f. the total number of parameters counting intercepts, AIC, p the number of regression coefficients, LR chi^2 the likelihood ratio chi-square statistic for testing the predictors, LR - p a chance-corrected LR chi-square, LR chi^2 test for PO the likelihood ratio chi-square test statistic for testing the PO assumption (by comparing -2 log likelihood for a relaxed model to that of a fully PO model), d.f. the degrees of freedom for this test, Pr(>chi^2) the P-value for this test, MCS R2 the Maddala-Cox-Snell R2 using the actual sample size, MCS R2 adj (MCS R2 adjusted for estimating p regression coefficients by subtracting p from LR), McFadden R2, McFadden R2 adj (an AIC-like adjustment proposed by McFadden without full justification), Mean |difference} from PO the overall mean absolute difference between predicted probabilities over all categories of Y and over all covariate settings. mad contains newdata and separately by rows in newdata the mean absolute difference (over Y categories) between estimated probabilities by the indicated relaxed model and those from the PO model. nboot is the number of successful bootstrap repetitions, and boot is a 4-way array with dimensions represented by the nboot resamples, the number of rows in newdata, the number of outcome levels, and elements for PPO and multinomial. For the modifications of the Maddala-Cox-Snell indexes see Hmisc::R2Measures.

Author(s)

Frank Harrell fh@fharrell.com

References

Adjusted R-square note

See Also

nnet::multinom(), VGAM::vglm(), lrm(), Hmisc::propsPO(), Hmisc::R2Measures(), Hmisc::combine.levels()

Examples

```
## Not run:
set.seed(1)
age <- rnorm(500, 50, 10)
sex <- sample(c('female', 'male'), 500, TRUE)
y   <- sample(0:4, 500, TRUE)
d   <- expand.grid(age=50, sex=c('female', 'male'))
w   <- impactPO(y ~ age + sex, nonpo = ~ sex, newdata=d)
w
# Note that PO model is a better model than multinomial (lower AIC)
# since multinomial model's improvement in fit is low in comparison
# with number of additional parameters estimated.  Same for PO model
# in comparison with partial PO model.

# Reverse levels of y so stacked bars have higher y located higher
```

```
revo <- function(z) {
  z <- as.factor(z)
  factor(z, levels=rev(levels(as.factor(z))))
}

require(ggplot2)
ggplot(w$estimates, aes(x=method, y=Probability, fill=revo(y))) +
  facet_wrap(~ sex) + geom_col() +
  xlab('') + guides(fill=guide_legend(title=''))

# Now vary 2 predictors

d <- expand.grid(sex=c('female', 'male'), age=c(40, 60))
w <- impactPO(y ~ age + sex, nonpo = ~ sex, newdata=d)
w
ggplot(w$estimates, aes(x=method, y=Probability, fill=revo(y))) +
  facet_grid(age ~ sex) + geom_col() +
 xlab('') + guides(fill=guide_legend(title=''))

## End(Not run)
```

`importedexported` *Exported Functions That Were Imported From Other Packages*

Description

`Surv` and `ggplot` are imported from, respectively, the `survival` and `ggplot2` packages and are exported from `rms` so that the user does not have to attach these packages to do simple things.

Usage

```
Surv(time, time2, event,
     type = c("right", "left", "interval", "counting", "interval2"),
     origin = 0)

ggplot(data = NULL, mapping = aes(), ..., environment =  parent.frame())
```

Arguments

`time`, `time2`, `event`, `type`, `origin`
: see `Surv`

`data`, `mapping`, `...`, `environment`
: see `ggplot`

Value

see documentation in the original packages

See Also

Surv, ggplot

Examples

```
## Not run:
f <- psm(Surv(dtime, death) ~ x1 + x2 + sex + race, dist='gau')
ggplot(Predict(f))

## End(Not run)
```

---

infoMxop *Operate on Information Matrices*

Description

Processes four types of information matrices: ones produced by the SparseM package for the orm or lrm functions in rms version 6.9-0 and earlier, by the Matrix package for version 7.0-0 of rms using a tri-band diagonal matrix for the intercepts, using Matrix for general sparse information matrices for intercepts (when any interval-censored observations exist), or plain matrices. For Matrix, the input information matrix is a list with three elements: a containing in two columns the diagonal and superdiagonal for intercepts (when there is no interval censoring) or a list with three elements row, col, a (when there is interval censoring), b, a square matrix for the covariates, and ab for intercepts x covariates. If nothing else is specified, the assembled information matrix is returned for Matrix, or the original info otherwise. If p=TRUE, the number of parameters in the model (number of rows and columns in the whole information matrix) is returned. If i is given, the i elements of the inverse of info are returned, using efficient calculation to avoid inverting the whole matrix. Otherwise if invert=TRUE or B is given without i, the efficiently (if Matrix or SparseM) inverted matrix is returned, or the matrix multiplication of the inverse and B. If both i and B are given, what is returned is the i portion of the inverse of the information matrix, matrix multiplied by B. This is done inside solve().

Usage

```
infoMxop(
  info,
  i,
  invert = !missing(i) || !missing(B),
  B,
  np = FALSE,
  tol = .Machine$double.eps,
  abort = TRUE
)
```

Arguments

`info`	an information matrix object
`i`	integer vector specifying elements returned from the inverse. You an also specify `i='x'` to return non-intercepts or `i='i'` to return intercepts.
`invert`	set to `TRUE` to invert `info` (implied when `i` or `B` is given)
`B`	multiplier matrix
`np`	set to `TRUE` to just fetch the total number of parameters (intercepts + betas)
`tol`	tolerance for matrix inversion singularity
`abort`	set to `FALSE` to run the `solve` calculation through `try()` without aborting; the user will detect that the operation did not success by examinine `inherits(result, 'try-error')` for being `TRUE`.

Details

When only variance-covariance matrix elements corresponding to the non-intercepts are desired, specify `i='x'` or `i=(k + 1) : nv` where `nv` is the number of intercepts and slopes combined. `infoMxop` computes the needed covariance matrix very quickly in this case. When inverting `info`, if `info` has a `'scale'` attribute with elements `mean` and `sd`, the scaling is reversed after inverting `info`.

When the number of intercepts is needed to be known and the `info` object is not a 3-element list, `info` must have an `intercepts` attribute to define the number. This is used for example when `transx=TRUE` is specified to `lrm` or `lrm.fit`.

Value

a single integer or a matrix

Author(s)

Frank Harrell

Examples

```
## Not run:
f <- orm(y ~ x)
infoMxop(f$info.matrix)   # assembles 3 pieces
infoMxop(v, i=c(2,4))     # returns a submatrix of v inverse
infoMxop(f$info.matrix, i='x')  # sub-covariance matrix for just the betas

## End(Not run)
```

intCalibration *Check Parallelism Assumption of Ordinal Semiparametric Models*

Description

For all the observations used a model fit, computes the estimated probability that Y is greater than each of a number of cutoffs, and compares this to smoothed estimated probabilities as a function of predicted probabilities, to obtain internal model calibration plots with multiple cutpoints. When Y is uncensored these are smoothed moving empirical cumulative distribution function estimates, and when Y has censored observations these are smoothing moving Kaplan-Meier estimates. `Hmisc::movStats()` is used to do the moving overlapping window calculations. When `hare=TRUE`, adaptive linear spline hazard regression estimates are also made, using `polspline::hare()`. When `ordsurv=TRUE`, adaptive ordinal regression estimates are made in addition.

Usage

```
intCalibration(
  fit,
  ycuts,
  m,
  x,
  onlydata = FALSE,
  eps = 25,
  bass = 9,
  tsmooth = "lowess",
  hare = TRUE,
  ordsurv = TRUE,
  dec = 4,
  xlab = bquote(hat(P)(.(yname) > y)),
  ylab = "Nonparametric Estimate",
  nrow = 1,
  ...
)
```

Arguments

`fit`	a fit object for which there is a `survest()` method, with `x=TRUE, y=TRUE` in effect
`ycuts`	a vector of cutpoints on Y
`m`	used when `ycuts` is not given. The lowest cutoff is chosen as the first Y value having at meast `m` uncensored observations to its left, and the highest cutoff is chosen so that there are at least `m` uncensored observations to the right of it. Cutoffs are equally spaced between these values in terms of number of uncensored observations. If omitted, `m` is set to the minimum of 50 and one quarter of the uncensored sample size.

x	a variable for which calibration-in-the-small is desired, instead of plotting predicted vs. observed probabilities. x will typically be chosen by virtue of being a strong predictor (such that lack of fit will matter more) but doesn't have to be in the model.
onlydata	set to TRUE to return a data frame suitable for plotting instead of actually plotting
eps, bass, tsmooth, hare, ordsurv	see Hmisc::movStats()
dec	number of digits to the right of the decimal place to which to round computed ycuts
xlab	x-axis label with default constructed from the Y-variable name in the model fit (y-axis label when x is specified)
ylab	y-axis label
nrow	if hare=TRUE or ordsurv=TRUE, the number of rows in the graph (must be 1 or 2)
...	other arguments passed to Hmisc::movStats(). To control the number of knots for ordsurv=TRUE specify k= here.

Details

These plots are plots of calibration-in-the-small. Alternate calibration-in-the-small plots may be obtained by specifying a predictor variable x against which to plot both predicted and observed probabilties as a function of x. This is the only place in the rms package where the "total effect" of a predictor is estimated instead of a partial effect. When x varies and moving overlapping windows of predicted and observed exceedance probabilities are estimated, if x is collinear with other predictors, they will "come along for the ride".

The function also prints information on calibration-in-the-large, i.e., the mean predicted probability of being beyond each cutpoint vs. the overall proportion of observations above that cutpoint. This is when x is not given.

Value

ggplot2 object or a data frame

Author(s)

Frank Harrell

Examples

```
## Not run:
getHdata(nhgh)
f <- orm(gh ~ rcs(age, 4), data=nhgh, family='loglog', x=TRUE, y=TRUE)
intCalibration(f, ycuts=c(5, 5.5, 6, 6.5))
f <- update(f, family='cloglog')
intCalibration(f, ycuts=c(5, 5.5, 6, 6.5))
intCalibration(f, ycuts=c(5, 6, 7), x=nhgh$age)

## End(Not run)
```

is.na.Ocens — *is.na Method for Ocens Objects*

Description

is.na Method for Ocens Objects

Usage

```
## S3 method for class 'Ocens'
is.na(x)
```

Arguments

x	an object created by Ocens

Value

a logical vector whose length is the number of rows in x, with TRUE designating observations having one or both columns of x equal to NA

Examples

```
Y <- Ocens(c(1, 2, NA, 4))
Y
is.na(Y)
```

latex.cph — *LaTeX Representation of a Fitted Cox Model*

Description

Creates a file containing a LaTeX representation of the fitted model.

Usage

```
## S3 method for class 'cph'
latex(object, title,
      file='',
      append=FALSE, surv=TRUE, maxt=FALSE, which=NULL, varnames, columns=65,
      inline=FALSE, before=if(inline)"" else "& &", after="", dec=3,
      pretrans=TRUE, caption, digits=.Options$digits, size="",
      ...) # for cph fit

## S3 method for class 'lrm'
latex(object, title, file, append, which, varnames,
```

```
columns, inline, before, after, pretrans, caption,
digits=.Options$digits, size="", ...) # for lrm fit

## S3 method for class 'ols'
latex(object, title, file, append, which, varnames,
columns, inline, before, after, pretrans, caption,
digits=.Options$digits, size="", ...) # ols fit

## S3 method for class 'orm'
latex(object, title, file, append, which, varnames,
columns, inline, before, after, pretrans, caption,
digits=.Options$digits, size="", intercepts=nrp < 10, ...) # for orm fit

## S3 method for class 'pphsm'
latex(object, title, file, append, which=NULL, varnames,
columns, inline, before, after, pretrans, caption,
digits=.Options$digits, size="", ...) # pphsm fit

## S3 method for class 'psm'
latex(object, title, file, append, which=NULL, varnames,
columns, inline, before, after, pretrans, caption,
digits=.Options$digits, size="", ...) # psm fit
```

Arguments

object	a fit object created by a rms fitting function.
title	ignored
file, append	see latex.default. Defaults to the console. When using html/markdown, file is ignored.
surv	if surv=TRUE was specified to cph, the underlying survival probabilities from object$surv.summary will be placed in a table unless surv=FALSE.
maxt	if the maximum follow-up time in the data (object$maxtime) exceeds the last entry in object$surv.summary, underlying survival estimates at object$maxtime will be added to the table if maxt=TRUE.
which, varnames, columns, inline, before, dec, pretrans	see latex.default
after	if not an empty string, added to end of markup if inline=TRUE
caption	a character string specifying a title for the equation to be centered and typeset in bold face. Default is no title.
digits	see latexrms
size	a LaTeX size to use, without the slash. Default is the prevailing size
intercepts	for orm fits. Default is to print intercepts if they are fewer than 10 in number. Set to TRUE or FALSE to force.
...	ignored

Value

the name of the created file, with class `c("latex","file")`. This object works with latex viewing and printing commands in Hmisc. If `file=''` and `options(prType=x` is in effect, where `x` is `"html"`, `"markdown"` or `"md"`, the result is run through `knitr::asis_output` so that it will be rendered correctly no matter which options are in effect in the chunk header.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

`latexrms`, `rcspline.restate`, `latex`

Examples

```
## Not run:
require(survival)
units(ftime) <- "Day"
f <- cph(Surv(ftime, death) ~ rcs(age)+sex, surv=TRUE, time.inc=60)
w <- latex(f, file='f.tex')  #Interprets fitted model and makes table of S0(t)
               #for t=0,60,120,180,...
w              #displays image, if viewer installed and file given above
latex(f)   # send LaTeX code to the console for knitr
options(prType='html')
latex(f)       # for use with knitr and R Markdown/Quarto using MathJax

## End(Not run)
```

---

latexrms — *LaTeX Representation of a Fitted Model*

Description

Creates a file containing a LaTeX representation of the fitted model. For model-specific typesetting there is `latex.lrm`, `latex.cph`, `latex.psm` and `latex.ols`. `latex.cph` has some arguments that are specific to `cph` models. `latexrms` is the core function which is called internally by `latexrms` (which is called by `latex.cph`, `latex.ols`, etc.). `html` and R Markdown-compatible markup (using MathJax) are written if `options(prType='html')`.

Usage

```
latexrms(object,
 file='',
 append=FALSE, which=1:p, varnames, columns=65, prefix=NULL, inline=FALSE,
 before=if(inline)"" else "& &", after="", intercept, pretrans=TRUE,
 digits=.Options$digits, size="")
```

Arguments

object	a fit object created by a fitting function in the rms series
file	name of .tex file to create, default is to write to console. file is ignored when options(prType='html'.
append	whether or not to append to an existing file
which	a vector of subcripts (corresponding to object$Design$name) specifying a submodel to print. Default is to describe the whole model. which can also be a vector of character strings specifying the factor names to print. Enough of each string is needed to ensure a unique match. Names for interaction effects are of the form "age * sex". For any interaction effect for which you do not request main effects, the main effects will be added to which. When which is given, the model structural statement is not included. In this case, intercepts are not included either.
varnames	variable names to substitute for non-interactions. Order must correspond to object$Design$name and interactions must be omitted. Default is object$Design$name[object$Desig varnames can contain any LaTeX commands such as subscripts and "\\\\frac" (all "\" must be quadrupled.) Any "/" must be preceeded by "\\" (2, not 4 backslashes). Elements of varnames for interactions are ignored; they can be set to any value.
columns	maximum number of columns of printing characters to allow before outputting a LaTeX newline command
prefix	if given, a LaTeX \lefteqn command of the form \lefteqn{prefix =} \\ will be inserted to print a left-hand-side of the equation.
inline	Set to TRUE to create text for insertion in an in-line equation. This text contains only the expansion of X beta, and is not surrounded by "$".
before	a character string to place before each line of output. Use the default for a LaTeX eqnarray environment. For inline=TRUE, the before string, if not an empty string, will be placed once before the entire markup.
after	a character string to place after the output if inline=TRUE
intercept	a special intercept value to include that is not part of the standard model parameters (e.g., centering constant in Cox model). Only allowed in the latexrms rendition.
pretrans	if any spline or polynomial-expanded variables are themselves transformed, a table of pre-transformations will be formed unless pretrans=FALSE.
digits	number of digits of precision to use in formatting coefficients and other numbers
size	a LaTeX font size to use for the output, without the slash. Default is current size.

Value

latexrms returns a character vector if file='', otherwise writes the output to file. For particular model fits, the latex method returns the result of running knitr::asis_output on the LaTeX or HTML code if file='', options(prType) was set but not to 'plain', and if knitr is currently running. This causes correct output to be rendered whether or not results='asis' appeared in the R Markdown or Quarto chunk header.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

latex, rcspline.restate, rms

Examples

```
## Not run:
f <- lrm(death ~ rcs(age)+sex)
w <- latex(f, file='f.tex')
w     # displays, using e.g. xdvi
latex(f)    # send LaTeX code to console, as for knitr
options(prType='html')
latex(f)    # emit html and latex for knitr html and html notebooks

## End(Not run)
```

---

lrm — *Logistic Regression Model*

Description

Fit binary and proportional odds ordinal logistic regression models using maximum likelihood estimation or penalized maximum likelihood estimation. See `cr.setup` for how to fit forward continuation ratio models with `lrm`. The fitting function used by `lrm` is `lrm.fit`, for which details and comparisons of its various optimization methods may be found here.

For the `print` method, format of output is controlled by the user previously running `options(prType="lang")` where `lang` is `"plain"` (the default), `"latex"`, or `"html"`. When using html with Quarto or RMarkdown, `results='asis'` need not be written in the chunk header.

Usage

```
lrm(formula, data=environment(formula),
    subset, na.action=na.delete, method="lrm.fit",
    model=FALSE, x=FALSE, y=FALSE, linear.predictors=TRUE, se.fit=FALSE,
    penalty=0, penalty.matrix,
    var.penalty,
    weights, normwt=FALSE, scale, ...)

## S3 method for class 'lrm'
print(x, digits=4, r2=c(0,2,4),
    coefs=TRUE, pg=FALSE,
    intercepts=x$non.slopes < 10,
    title='Logistic Regression Model', ...)
```

Arguments

`formula` a formula object. An `offset` term can be included. The offset causes fitting of a model such as $logit(Y = 1) = X\beta + W$, where W is the offset variable having no estimated coefficient. The response variable can be any data type; `lrm` converts it in alphabetic or numeric order to an S factor variable and recodes it 0,1,2,... internally.

`data` data frame to use. Default is the current frame.

`subset` logical expression or vector of subscripts defining a subset of observations to analyze

`na.action` function to handle `NA`s in the data. Default is `na.delete`, which deletes any observation having response or predictor missing, while preserving the attributes of the predictors and maintaining frequencies of deletions due to each variable in the model. This is usually specified using `options(na.action="na.delete")`.

`method` name of fitting function. Only allowable choice at present is `lrm.fit`.

`model` causes the model frame to be returned in the fit object

`x` causes the expanded design matrix (with missings excluded) to be returned under the name `x`. For `print`, an object created by `lrm`.

`y` causes the response variable (with missings excluded) to be returned under the name `y`.

`linear.predictors` causes the predicted X beta (with missings excluded) to be returned under the name `linear.predictors`. When the response variable has more than two levels, the first intercept is used.

`se.fit` causes the standard errors of the fitted values to be returned under the name `se.fit`.

`penalty` The penalty factor subtracted from the log likelihood is $0.5\beta' P\beta$, where β is the vector of regression coefficients other than intercept(s), and P is `penalty factors * penalty.matrix` and `penalty.matrix` is defined below. The default is `penalty=0` implying that ordinary unpenalized maximum likelihood estimation is used. If `penalty` is a scalar, it is assumed to be a penalty factor that applies to all non-intercept parameters in the model. Alternatively, specify a list to penalize different types of model terms by differing amounts. The elements in this list are named `simple`, `nonlinear`, `interaction` and `nonlinear.interaction`. If you omit elements on the right of this series, values are inherited from elements on the left. Examples: `penalty=list(simple=5, nonlinear=10)` uses a penalty factor of 10 for nonlinear or interaction terms. `penalty=list(simple=0, nonlinear=2, nonlinear.interaction=4)` does not penalize linear main effects, uses a penalty factor of 2 for nonlinear or interaction effects (that are not both), and 4 for nonlinear interaction effects.

`penalty.matrix` specifies the symmetric penalty matrix for non-intercept terms. The default matrix for continuous predictors has the variance of the columns of the design matrix in its diagonal elements so that the penalty to the log likelihood is unitless. For main effects for categorical predictors with c categories, the rows and columns of the matrix contain a $c - 1 \times c - 1$ sub-matrix that is used to compute the sum of squares about the mean of the c parameter values (setting

the parameter to zero for the reference cell) as the penalty component for that predictor. This makes the penalty independent of the choice of the reference cell. If you specify `penalty.matrix`, you may set the rows and columns for certain parameters to zero so as to not penalize those parameters. Depending on `penalty`, some elements of `penalty.matrix` may be overridden automatically by setting them to zero. The penalty matrix that is used in the actual fit is $penalty \times diag(pf) \times penalty.matrix \times diag(pf)$, where pf is the vector of square roots of penalty factors computed from `penalty` by `Penalty.setup` in `rmsMisc`. If you specify `penalty.matrix` you must specify a nonzero value of `penalty` or no penalization will be done.

`var.penalty` deprecated and ignored

`weights` a vector (same length as `y`) of possibly fractional case weights

`normwt` set to `TRUE` to scale `weights` so they sum to the length of `y`; useful for sample surveys as opposed to the default of frequency weighting

`scale` deprecated; see `lrm.fit` `transx` argument

`...` arguments that are passed to `lrm.fit`, or from `print`, to `prModFit`

`digits` number of significant digits to use

`r2` vector of integers specifying which R^2 measures to print, with 0 for Nagelkerke R^2 and 1:4 corresponding to the 4 measures computed by `R2Measures`. Default is to print Nagelkerke (labeled R2) and second and fourth `R2Measures` which are the measures adjusted for the number of predictors, first for the raw sample size then for the effective sample size, which here is from the formula for the approximate variance of a log odds ratio in a proportional odds model.

`coefs` specify `coefs=FALSE` to suppress printing the table of model coefficients, standard errors, etc. Specify `coefs=n` to print only the first `n` regression coefficients in the model.

`pg` set to `TRUE` to print g-indexes

`intercepts` controls printing of intercepts. By default they are only printed if there aren't more than 10 of them.

`title` a character string title to be passed to `prModFit`

Value

The returned fit object of `lrm` contains the following components in addition to the ones mentioned under the optional arguments.

`call` calling expression

`freq` table of frequencies for `Y` in order of increasing `Y`

`stats` vector with the following elements: number of observations used in the fit, maximum absolute value of first derivative of log likelihood, model likelihood ratio χ^2, d.f., P-value, c index (area under ROC curve), Somers' D_{xy}, Goodman-Kruskal γ, Kendall's τ_a rank correlations between predicted probabilities and observed response, the Nagelkerke R^2 index, the Brier score computed with respect to $Y >$ its lowest level, the g-index, gr (the g-index on the odds ratio scale), and gp (the g-index on the probability scale using the same cutoff used for

the Brier score). Probabilities are rounded to the nearest 0.0002 in the computations or rank correlation indexes. In the case of penalized estimation, the `"Model L.R."` is computed without the penalty factor, and `"d.f."` is the effective d.f. from Gray's (1992) Equation 2.9. The P-value uses this corrected model L.R. χ^2 and corrected d.f. The score chi-square statistic uses first derivatives which contain penalty components.

`fail` set to `TRUE` if convergence failed (and `maxiter>1`)

`coefficients` estimated parameters

`var` estimated variance-covariance matrix (inverse of information matrix). If `penalty>0`, `var` is either the inverse of the penalized information matrix.

`effective.df.diagonal`
is returned if `penalty>0`. It is the vector whose sum is the effective d.f. of the model (counting intercept terms).

`u` vector of first derivatives of log-likelihood

`deviance` -2 log likelihoods (counting penalty components) When an offset variable is present, three deviances are computed: for intercept(s) only, for intercepts+offset, and for intercepts+offset+predictors. When there is no offset variable, the vector contains deviances for the intercept(s)-only model and the model with intercept(s) and predictors.

`est` vector of column numbers of `X` fitted (intercepts are not counted)

`non.slopes` number of intercepts in model

`penalty` see above

`penalty.matrix` the penalty matrix actually used in the estimation

Note

When creating ordinal levels for non-integer numeric Y, this function does not use the `unique` function because it was found to be non-reproducible across hardware platforms. Instead, intercepts are mapped to integers obtained from rounding original Y levels after multiplying by 1e7 (by default). The only way to guarantee the correct mapping to intercepts is to reference the "official" levels of Y stored in the `yunique` object found in the fit object. Users may opt instead to replace original data with e.g. 1e-7 * round(Y * 1e7) before model fitting.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Le Cessie S, Van Houwelingen JC: Ridge estimators in logistic regression. Applied Statistics 41:191–201, 1992.

Verweij PJM, Van Houwelingen JC: Penalized likelihood in Cox regression. Stat in Med 13:2427–2436, 1994.

Gray RJ: Flexible methods for analyzing survival data using splines, with applications to breast cancer prognosis. JASA 87:942–951, 1992.

Shao J: Linear model selection by cross-validation. JASA 88:486–494, 1993.

Verweij PJM, Van Houwelingen JC: Crossvalidation in survival analysis. Stat in Med 12:2305–2314, 1993.

Harrell FE: Model uncertainty, penalization, and parsimony. ISCB Presentation on UVa Web page, 1998.

See Also

lrm.fit, predict.lrm, rms.trans, rms, glm, latex.lrm, residuals.lrm, na.delete, na.detail.response, pentrace, rmsMisc, vif, cr.setup, predab.resample, validate.lrm, calibrate, Mean.lrm, gIndex, prModFit

Examples

```
#Fit a logistic model containing predictors age, blood.pressure, sex
#and cholesterol, with age fitted with a smooth 5-knot restricted cubic
#spline function and a different shape of the age relationship for males
#and females.  As an intermediate step, predict mean cholesterol from
#age using a proportional odds ordinal logistic model
#
require(ggplot2)
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
label(age)            <- 'Age'      # label is in Hmisc
label(cholesterol)    <- 'Total Cholesterol'
label(blood.pressure) <- 'Systolic Blood Pressure'
label(sex)            <- 'Sex'
units(cholesterol)    <- 'mg/dl'   # uses units.default in Hmisc
units(blood.pressure) <- 'mmHg'

# Group cholesterol unnecessarily into 40-tiles
ch <- cut2(cholesterol, g=40, levels.mean=TRUE) # use mean values in intervals
table(ch)
f <- lrm(ch ~ age)
# options(prType='latex')
print(f)  # write latex code to console if prType='latex' is in effect
m <- Mean(f)    # see help file for Mean.lrm
d <- data.frame(age=seq(0,90,by=10))
m(predict(f, d))
# Repeat using ols
f <- ols(cholesterol ~ age)
predict(f, d)

# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
```

```
     (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male'))
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)
cholesterol[1:3] <- NA   # 3 missings, at random

ddist <- datadist(age, blood.pressure, cholesterol, sex)
options(datadist='ddist')

fit <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)),
               x=TRUE, y=TRUE)
#      x=TRUE, y=TRUE allows use of resid(), which.influence below
#      could define d <- datadist(fit) after lrm(), but data distribution
#      summary would not be stored with fit, so later uses of Predict
#      or summary.rms would require access to the original dataset or
#      d or specifying all variable values to summary, Predict, nomogram
anova(fit)
p <- Predict(fit, age, sex)
ggplot(p)   # or plot()
ggplot(Predict(fit, age=20:70, sex="male"))   # need if datadist not used
print(cbind(resid(fit,"dfbetas"), resid(fit,"dffits"))[1:20,])
which.influence(fit, .3)
# latex(fit)                      #print nice statement of fitted model
#
#Repeat this fit using penalized MLE, penalizing complex terms
#(for nonlinear or interaction effects)
#
fitp <- update(fit, penalty=list(simple=0,nonlinear=10), x=TRUE, y=TRUE)
effective.df(fitp)
# or lrm(y ~ \dots, penalty=\dots)


#Get fits for a variety of penalties and assess predictive accuracy
#in a new data set.  Program efficiently so that complex design
#matrices are only created once.


set.seed(201)
x1 <- rnorm(500)
x2 <- rnorm(500)
x3 <- sample(0:1,500,rep=TRUE)
L  <- x1+abs(x2)+x3
y  <- ifelse(runif(500)<=plogis(L), 1, 0)
new.data <- data.frame(x1,x2,x3,y)[301:500,]
#
for(penlty in seq(0,.15,by=.005)) {
  if(penlty==0) {
    f <- lrm(y ~ rcs(x1,4)+rcs(x2,6)*x3, subset=1:300, x=TRUE, y=TRUE)
    # True model is linear in x1 and has no interaction
    X <- f$x    # saves time for future runs - don't have to use rcs etc.
    Y <- f$y    # this also deletes rows with NAs (if there were any)
    penalty.matrix <- diag(diag(var(X)))
    Xnew <- predict(f, new.data, type="x")
    # expand design matrix for new data
```

```
    Ynew <- new.data$y
  } else f <- lrm.fit(X,Y, penalty.matrix=penlty*penalty.matrix)
#
  cat("\nPenalty :",penlty,"\n")
  pred.logit <- f$coef[1] + (Xnew %*% f$coef[-1])
  pred <- plogis(pred.logit)
  C.index <- somers2(pred, Ynew)["C"]
  Brier   <- mean((pred-Ynew)^2)
  Deviance<- -2*sum( Ynew*log(pred) + (1-Ynew)*log(1-pred) )
  cat("ROC area:",format(C.index),"   Brier score:",format(Brier),
      "   -2 Log L:",format(Deviance),"\n")
}
#penalty=0.045 gave lowest -2 Log L, Brier, ROC in test sample for S+
#
#Use bootstrap validation to estimate predictive accuracy of
#logistic models with various penalties
#To see how noisy cross-validation estimates can be, change the
#validate(f, \dots) to validate(f, method="cross", B=10) for example.
#You will see tremendous variation in accuracy with minute changes in
#the penalty.  This comes from the error inherent in using 10-fold
#cross validation but also because we are not fixing the splits.
#20-fold cross validation was even worse for some
#indexes because of the small test sample size.  Stability would be
#obtained by using the same sample splits for all penalty values
#(see above), but then we wouldn't be sure that the choice of the
#best penalty is not specific to how the sample was split.  This
#problem is addressed in the last example.
#
penalties <- seq(0,.7,length=3)   # really use by=.02
index <- matrix(NA, nrow=length(penalties), ncol=11,
        dimnames=list(format(penalties),
          c("Dxy","R2","Intercept","Slope","Emax","D","U","Q","B","g","gp")))
i <- 0
for(penlty in penalties)
{
  cat(penlty, "")
  i <- i+1
  if(penlty==0)
    {
    f <- lrm(y ~ rcs(x1,4)+rcs(x2,6)*x3, x=TRUE, y=TRUE)  # fit whole sample
    X <- f$x
    Y <- f$y
    penalty.matrix <- diag(diag(var(X)))   # save time - only do once
    }
  else
   f <- lrm(Y ~ X, penalty=penlty,
            penalty.matrix=penalty.matrix, x=TRUE,y=TRUE)
  val <- validate(f, method="boot", B=20)  # use larger B in practice
  index[i,] <- val[,"index.corrected"]
}
par(mfrow=c(3,3))
for(i in 1:9)
{
```

```
  plot(penalties, index[,i],
       xlab="Penalty", ylab=dimnames(index)[[2]][i])
  lines(lowess(penalties, index[,i]))
}
options(datadist=NULL)

# Example of weighted analysis
x    <- 1:5
y    <- c(0,1,0,1,0)
reps <- c(1,2,3,2,1)
lrm(y ~ x, weights=reps)
x <- rep(x, reps)
y <- rep(y, reps)
lrm(y ~ x)   # same as above

#
#Study performance of a modified AIC which uses the effective d.f.
#See Verweij and Van Houwelingen (1994) Eq. (6).  Here AIC=chisq-2*df.
#Also try as effective d.f. equation (4) of the previous reference.
#Also study performance of Shao's cross-validation technique (which was
#designed to pick the "right" set of variables, and uses a much smaller
#training sample than most methods).  Compare cross-validated deviance
#vs. penalty to the gold standard accuracy on a 7500 observation dataset.
#Note that if you only want to get AIC or Schwarz Bayesian information
#criterion, all you need is to invoke the pentrace function.
#NOTE: the effective.df( ) function is used in practice
#
## Not run:
for(seed in c(339,777,22,111,3)){
# study performance for several datasets
  set.seed(seed)
  n <- 175; p <- 8
  X <- matrix(rnorm(n*p), ncol=p) # p normal(0,1) predictors
  Coef <- c(-.1,.2,-.3,.4,-.5,.6,-.65,.7)  # true population coefficients
  L <- X %*% Coef                 # intercept is zero
  Y <- ifelse(runif(n)<=plogis(L), 1, 0)
  pm <- diag(diag(var(X)))
  #Generate a large validation sample to use as a gold standard
  n.val <- 7500
  X.val <- matrix(rnorm(n.val*p), ncol=p)
  L.val <- X.val %*% Coef
  Y.val <- ifelse(runif(n.val)<=plogis(L.val), 1, 0)
  #
  Penalty <- seq(0,30,by=1)
  reps <- length(Penalty)
  effective.df <- effective.df2 <- aic <- aic2 <- deviance.val <-
    Lpenalty <- single(reps)
  n.t <- round(n^.75)
  ncv <- c(10,20,30,40)     # try various no. of reps in cross-val.
  deviance <- matrix(NA,nrow=reps,ncol=length(ncv))
  #If model were complex, could have started things off by getting X, Y
  #penalty.matrix from an initial lrm fit to save time
  #
```

```
  for(i in 1:reps) {
    pen <- Penalty[i]
    cat(format(pen),"")
    f.full <- lrm.fit(X, Y, penalty.matrix=pen*pm)
    Lpenalty[i] <- pen* t(f.full$coef[-1]) %*% pm %*% f.full$coef[-1]
    f.full.nopenalty <- lrm.fit(X, Y, initial=f.full$coef, maxit=1)
    info.matrix.unpenalized <- solve(f.full.nopenalty$var)
    effective.df[i] <- sum(diag(info.matrix.unpenalized %*% f.full$var)) - 1
    lrchisq <- f.full.nopenalty$stats["Model L.R."]
    # lrm does all this penalty adjustment automatically (for var, d.f.,
    # chi-square)
    aic[i] <- lrchisq - 2*effective.df[i]
    #
    pred <- plogis(f.full$linear.predictors)
    score.matrix <- cbind(1,X) * (Y - pred)
    sum.u.uprime <- t(score.matrix) %*% score.matrix
    effective.df2[i] <- sum(diag(f.full$var %*% sum.u.uprime))
    aic2[i] <- lrchisq - 2*effective.df2[i]
    #
    #Shao suggested averaging 2*n cross-validations, but let's do only 40
    #and stop along the way to see if fewer is OK
    dev <- 0
    for(j in 1:max(ncv)) {
      s    <- sample(1:n, n.t)
      cof  <- lrm.fit(X[s,],Y[s],
                      penalty.matrix=pen*pm)$coef
      pred <- cof[1] + (X[-s,] %*% cof[-1])
      dev <- dev -2*sum(Y[-s]*pred + log(1-plogis(pred)))
      for(k in 1:length(ncv)) if(j==ncv[k]) deviance[i,k] <- dev/j
    }
    #
    pred.val <- f.full$coef[1] + (X.val %*% f.full$coef[-1])
    prob.val <- plogis(pred.val)
    deviance.val[i] <- -2*sum(Y.val*pred.val + log(1-prob.val))
  }
  postscript(hor=TRUE)   # along with graphics.off() below, allow plots
  par(mfrow=c(2,4))   # to be printed as they are finished
  plot(Penalty, effective.df, type="l")
  lines(Penalty, effective.df2, lty=2)
  plot(Penalty, Lpenalty, type="l")
  title("Penalty on -2 log L")
  plot(Penalty, aic, type="l")
  lines(Penalty, aic2, lty=2)
  for(k in 1:length(ncv)) {
    plot(Penalty, deviance[,k], ylab="deviance")
    title(paste(ncv[k],"reps"))
    lines(supsmu(Penalty, deviance[,k]))
  }
  plot(Penalty, deviance.val, type="l")
  title("Gold Standard (n=7500)")
  title(sub=format(seed),adj=1,cex=.5)
  graphics.off()
}
```

```
## End(Not run)
#The results showed that to obtain a clear picture of the penalty-
#accuracy relationship one needs 30 or 40 reps in the cross-validation.
#For 4 of 5 samples, though, the super smoother was able to detect
#an accurate penalty giving the best (lowest) deviance using 10-fold
#cross-validation.  Cross-validation would have worked better had
#the same splits been used for all penalties.
#The AIC methods worked just as well and are much quicker to compute.
#The first AIC based on the effective d.f. in Gray's Eq. 2.9
#(Verweij and Van Houwelingen (1994) Eq. 5 (note typo)) worked best.
```

`lrm.fit` *lrm.fit*

Description

Logistic Model Fitter

Usage

```
lrm.fit(
  x,
  y,
  offset = 0,
  initial,
  opt_method = c("NR", "nlminb", "LM", "glm.fit", "nlm", "BFGS", "L-BFGS-B", "CG",
    "Nelder-Mead"),
  maxit = 50,
  reltol = 1e-10,
  abstol = if (opt_method %in% c("NR", "LM")) 1e+10 else 0,
  gradtol = if (opt_method %in% c("NR", "LM")) 0.001 else 1e-05,
  factr = 1e+07,
  eps = 5e-04,
  minstepsize = 0.01,
  trace = 0,
  tol = .Machine$double.eps,
  penalty.matrix = NULL,
  weights = NULL,
  normwt = FALSE,
  transx = FALSE,
  compstats = TRUE,
  inclpen = TRUE,
  initglm = FALSE,
  y.precision = 7
)
```

Arguments

x	design matrix with no column for an intercept. If a vector is transformed to a one-column matrix.
y	response vector, numeric, categorical, or character. For ordinal regression, the order of categories comes from `factor` levels, and if `y` is not a factor, from the numerical or alphabetic order of `y` values.
offset	optional numeric vector containing an offset on the logit scale
initial	vector of initial parameter estimates, beginning with the intercepts
opt_method	optimization method, with possible values • `'NR'` : the default, standard Newton-Raphson iteration using the gradient and Hessian, with step-helving. All three convergence criteria of `eps, gradtol, abstol` must be satisfied. Relax some of these if you do not want to consider some of them at all in judging convergence. The defaults for the various tolerances for `NR` result in convergence being mainly judged by `eps` in most uses. Tighten the non-`eps` parameters to give more weight to the other criteria. • `'LM'` : the Levenberg-Marquardt method, with the same convergence criteria as `'NR'` • `'nlminb'` : a quasi-Newton method using `stats::nlminb()` which uses gradients and the Hessian. This is a fast and robust algorithm. • `'glm.fit'` : for binary `y` without penalization only • `'nlm'` : see `stats::nlm()`; not highly recommended • `'BFGS'` : • `'L-BFGS-B'` : • `'CG'` : • `'Nelder-Mead'` : see `stats::optim()` for these 4 methods
maxit	maximum number of iterations allowed, which means different things for different `opt_method`. For `NR` it is the number of updates to parameters not counting step-halving steps. When `maxit=1`, `initial` is assumed to contain the maximum likelihood estimates already, and those are returned as `coefficients`, along with `u`, `info.matrix` (negative Hessian) and `deviance`. `stats` are only computed if `compstats` is explicitly set to `TRUE` by the user.
reltol	used by `BFGS`, `nlminb`, `glm.fit` to specify the convergence criteria in relative terms with regard to -2 LL, i.e., convergence is assume when one minus the fold-change falls below `reltol`
abstol	used by `NR` (maximum absolute change in parameter estimates from one iteration to the next before convergence can be declared; by default has no effect), `nlminb` (by default has no effect; see `abs.tol` argument; set to e.g. 0.001 for `nlminb` when there is complete separation)
gradtol	used by `NR` and `LM` (maximum absolute gradient before convergence can be declared) and `nlm` (similar but for a scaled gradient). For `NR` and `LM` `gradtol` is multiplied by the the sample size / 1000, because the gradient is proportional to sample size.
factr	see `stats::optim()` documentation for `L-BFGS-B`

eps	difference in -2 log likelihood for declaring convergence with `opt_method='NR'`. At present, the old `lrm.fit` approach of still declaring convergence even if the -2 LL gets worse by `eps/10` while the maximum absolute gradient is below 1e-9 is not implemented. This handles the case where the initial estimates are actually MLEs, and prevents endless step-halving.
minstepsize	used with `opt_method='NR'` to specify when to abandon step-halving
trace	set to a positive integer to trace the iterative process. Some optimization methods distinguish `trace=1` from `trace` higher than 1.
tol	QR singularity criterion for `opt_method='NR'` updates; ignored when inverting the final information matrix because `chol` is used for that.
penalty.matrix	a self-contained ready-to-use penalty matrix - see `lrm()`. It is pxp where p is the number of columns of `x`.
weights	a vector (same length as `y`) of possibly fractional case weights
normwt	set to `TRUE` to scale `weights` so they sum to n, the length of `y`; useful for sample surveys as opposed to the default of frequency weighting
transx	set to `TRUE` to center `x` and QR-factor it to orthogonalize. See this for details.
compstats	set to `FALSE` to prevent the calculation of the vector of model statistics
inclpen	set to `FALSE` to not include the penalty matrix in the Hessian when the Hessian is being computed on transformed `x`, vs. adding the penalty after back-transforming. This should not matter.
initglm	set to `TRUE` to compute starting values for an ordinal model by using `glm.fit` to fit a binary logistic model for predicting the probability that `y` exceeds or equals the median of `y`. After fitting the binary model, the usual starting estimates for intercepts (log odds of cumulative raw proportions) are all adjusted so that the intercept corresponding to the median is the one from `glm.fit`.
y.precision	when `y`` is numeric, values may need to be rounded to avoid unpredictable behavior with [to 7 decimal places.

Details

Fits a binary or propoortional odds ordinal logistic model for a given design matrix and response vector with no missing values in either. Ordinary or quadratic penalized maximum likelihood estimation is used.

`lrm.fit` implements a large number of optimization algorithms with the default being Newton-Raphson with step-halving. For binary logistic regression without penalization iteratively reweighted least squares method in `stats::glm.fit()` is an option. The -2 log likeilhood, gradient, and Hessian (negative information) matrix are computed in Fortran for speed. Optionally, the `x` matrix is mean-centered and QR-factored to help in optimization when there are strong collinearities. Parameter estimates and the covariance matrix are adjusted to the original `x` scale after fitting. More detail and comparisons of the various optimization methods may be found here. For ordinal regression with a large number of intercepts (distinct `y` values less one) you may want to use 'optim_method='BFGS', which does away with the need to compute the Hessian. This will be helpful if statistical tests and confidence intervals are not being computed, or when only likelihood ratio tests are done.

When using Newton-Raphson or Levenberg-Marquardt optimization, sparse Hessian/information/variance-covariance matrices are used throughout. For `nlminb` the Hessian has to be expanded into full non-sparse form, so `nlminb` will not be very efficient for a large number of intercepts.

When there is complete separation (Hauck-Donner condition), i.e., the MLE of a coefficient is $\pm\infty$, and `y` is binary and there is no penalty, `glm.fit` may not converge because it does not have a convergence parameter for the deviance. Setting `trace=1` will reveal that the -2LL is approaching zero but doesn't get there, relatively speaking. In such cases the default of `NR` with `eps=5e-4` or using `nlminb` with its default of `abstol=0.001` works well.

Value

a list with the following elements:

- `call`: the R call to `lrm.fit`
- `freq`: vector of `y` frequencies
- `ymedian`: median of original `y` values if `y` is numeric, otherwise the median of the integer-recorded version of `y`
- `yunique`: vector of distinct original `y` values, subject to rounding
- `sumty`: vector of weighted `y` frequencies
- `stats`: vector with a large number of indexes and model parameters (`NULL` if `compstats=FALSE`):
  - `Obs`: number of observations
  - `Max Deriv`: maximum absolute gradiant
  - `Model L.R.`: overall model LR chi-square statistic
  - `d.f.`: degrees of freedom (number of non-intercepts)
  - `P`: p-value for the overall `Model L.R.` and `d.f.`
  - `C`: concordance probability between predicted probability and `y`
  - `Dxy`: Somer's Dxy rank correlation between predicted probability and `y`, = 2(C - 0.5)
  - `Gamma`:
  - `Tau-a`:
  - `R2`: documented here; the first element, with the plain `'R2'` name is Nagelkerke's R^2
  - `Brier`: Brier score. For ordinal models this is computed with respect the the median intercept.
  - `g`: g-index (Gini's mean difference of linear predictors)
  - `gr`: g-index on the odds ratio scale
  - `gp`: g-index on the probability scale
- `fail`: `TRUE` if any matrix inversion or failure to converge occurred, `FALSE` otherwise
- `coefficients`:
- `info.matrix`: normally a list of 3 elements `a`, `b`, `ab` with `a` being a $k x 2$ matrix for k intercepts, `b` being $p x p$ for p predictors, and `ab` being $k x p$. See `infoMxop()` for easy ways of operating on these 3 elements. When `info.matrix` is not a 3-element list, as when `transx=TRUE`, it will have an `intercepts` attribute defining the number of intercepts in the model.
- `u`: gradient vector

- `iter`: number of iterations required. For some optimization methods this is a vector.
- `deviance`: vector of deviances: intercepts-only, intercepts + offset (if `offset` is present), final model (if `x` is used)
- `non.slopes`: number of intercepts in the model
- `linear.predictors`: vector of linear predictors at the median intercept
- `penalty.matrix`: penalty matrix or `NULL`
- `weights`: `weights` or `NULL`
- `xbar`: vector of column means of `x`, or `NULL` if `transx=FALSE`
- `xtrans`: input value of `transx`
- `R`: R matrix from QR to be used to rotate parameters back to original scale in the future
- `Ri`: inverse of `R`
- `opt_method`: input value

Author(s)

Frank Harrell fh@fharrell.com

See Also

`lrm()`, `stats::glm()`, `cr.setup()`, `gIndex()`, `stats::optim()`, `stats::nlminb()`, `stats::nlm()`,`stats::glm.fit()`, `recode2integer()`, `Hmisc::qrxcenter()`, `infoMxop()`

Examples

```
## Not run:
# Fit an additive logistic model containing numeric predictors age,
# blood.pressure, and sex, assumed to be already properly coded and
# transformed

fit <- lrm.fit(cbind(age,blood.pressure,sex=='male'), death)

## End(Not run)
```

`LRupdate` *LRupdate*

Description

Update Model LR Statistics After Multiple Imputation

Usage

```
LRupdate(fit, anova)
```

Arguments

fit	an rms fit object
anova	the result of processMI(..., 'anova')

Details

For fits from orm, lrm, orm, cph, psm that were created using fit.mult.impute with lrt=TRUE or equivalent options and for which anova was obtained using processMI(fit, 'anova') to compute imputation-adjusted LR statistics. LRupdate uses the last line of the anova result (containing the overall model LR chi-square) to update Model L.R. in the fit stats component, and to adjust any of the new R-square measures in stats.

For models using Nagelkerke's R-squared, these are set to NA as they would need to be recomputed with a new intercept-only log-likelihood, which is not computed by anova. For ols models, R-squared is left alone as it is sample-size-independent and print.ols prints the correct adjusted R-squared due to fit.mult.impute correcting the residual d.f. in stacked fits.

Value

new fit object like fit but with the substitutions made

Author(s)

Frank Harrell

See Also

processMI.fit.mult.impute(), Hmisc::R2Measures()

Examples

```
## Not run:
a <- aregImpute(~ y + x1 + x2, n.impute=30, data=d)
f <- fit.mult.impute(y ~ x1 + x2, lrm, a, data=d, lrt=TRUE)
a <- processMI(f, 'anova')
f <- LRupdate(f, a)
print(f, r2=1:4)   # print all imputation-corrected R2 measures

## End(Not run)
```

---

matinv — *Total and Partial Matrix Inversion using Gauss-Jordan Sweep Operator*

Description

This function inverts or partially inverts a matrix using pivoting (the sweep operator). It is useful for sequential model-building.

Usage

```
matinv(a, which, negate=TRUE, eps=1e-12)
```

Arguments

a	square matrix to invert or partially invert. May have been inverted or partially inverted previously by matinv, in which case its "swept" attribute is updated. Will un-invert if already inverted.
which	vector of column/row numbers in a to invert. Default is all, for total inverse.
negate	So that the algorithm can keep track of which pivots have been swept as well as roundoff errors, it actually returns the negative of the inverse or partial inverse. By default, these elements are negated to give the usual expected result. Set negate=FALSE if you will be passing the result right back into matinv, otherwise, negate the submatrix before sending back to matinv.
eps	singularity criterion

Value

a square matrix, with attributes "rank" and "swept".

References

Clarke MRB (1982). Algorithm AS 178: The Gauss-Jordan sweep operator with detection of collinearity. Appl Statist 31:166–9.

Ridout MS, Cobb JM (1986). Algorithm AS R78 : A remark on algorithm AS 178: The Gauss-Jordan sweep operator with detection of collinearity. Appl Statist 38:420–2.

See Also

lrm, solve

Examples

```
a      <- diag(1:3)
a.inv1 <- matinv(a, 1, negate=FALSE)      #Invert with respect to a[1,1]
a.inv1
a.inv  <- -matinv(a.inv1, 2:3, negate=FALSE) #Finish the job
a.inv
solve(a)
```

nomogram *Draw a Nomogram Representing a Regression Fit*

Description

Draws a partial nomogram that can be used to manually obtain predicted values from a regression model that was fitted with `rms`. The nomogram does not have lines representing sums, but it has a reference line for reading scoring points (default range 0–100). Once the reader manually totals the points, the predicted values can be read at the bottom. Non-monotonic transformations of continuous variables are handled (scales wrap around), as are transformations which have flat sections (tick marks are labeled with ranges). If interactions are in the model, one variable is picked as the "axis variable", and separate axes are constructed for each level of the interacting factors (preference is given automatically to using any discrete factors to construct separate axes) and levels of factors which are indirectly related to interacting factors (see DETAILS). Thus the nomogram is designed so that only one axis is actually read for each variable, since the variable combinations are disjoint. For categorical interacting factors, the default is to construct axes for all levels. The user may specify coordinates of each predictor to label on its axis, or use default values. If a factor interacts with other factors, settings for one or more of the interacting factors may be specified separately (this is mandatory for continuous variables). Optional confidence intervals will be drawn for individual scores as well as for the linear predictor. If more than one confidence level is chosen, multiple levels may be displayed using different colors or gray scales. Functions of the linear predictors may be added to the nomogram.

The `datadist` object that was in effect when the model was fit is used to specify the limits of the axis for continuous predictors when the user does not specify tick mark locations in the `nomogram` call.

`print.nomogram` prints axis information stored in an object returned by `nomogram`. This is useful in producing tables of point assignments by levels of predictors. It also prints how many linear predictor units there are per point and the number of points per unit change in the linear predictor.

`legend.nomabbrev` draws legends describing abbreviations used for labeling tick marks for levels of categorical predictors.

Usage

```
nomogram(fit, ..., adj.to, lp=TRUE, lp.at=NULL,
         fun=NULL, fun.at=NULL, fun.lp.at=NULL, funlabel="Predicted Value",
         interact=NULL, kint=NULL,  conf.int=FALSE,
         conf.lp=c("representative", "all", "none"),
         est.all=TRUE, posterior.summary=c('mean', 'median', 'mode'),
         abbrev=FALSE, minlength=4, maxscale=100, nint=10,
         vnames=c("labels","names"),
         varname.label=TRUE, varname.label.sep="=",
         omit=NULL, verbose=FALSE)

## S3 method for class 'nomogram'
print(x, dec=0, ...)
```

```
## S3 method for class 'nomogram'
plot(x, lplabel="Linear Predictor", fun.side,
 col.conf=c(1, 0.3),
 conf.space=c(.08,.2), label.every=1, force.label=FALSE,
 xfrac=.35, cex.axis=.85, cex.var=1, col.grid=NULL,
 varname.label=TRUE, varname.label.sep="=", ia.space=.7,
 tck=NA, tcl=-0.25, lmgp=.4, naxes,
 points.label='Points', total.points.label='Total Points',
 total.sep.page=FALSE, total.fun, cap.labels=FALSE, ...)

legend.nomabbrev(object, which, x, y, ncol=3, ...)
```

Arguments

`fit` a regression model fit that was created with `rms`, and (usually) with `options(datadist = "object.name")` in effect.

`...` settings of variables to use in constructing axes. If `datadist` was in effect, the default is to use `pretty(total range, nint)` for continuous variables, and the class levels for discrete ones. For `legend.nomabbrev`, `...` specifies optional parameters to pass to `legend`. Common ones are `bty = "n"` to suppress drawing the box. You may want to specify a non-proportionally spaced font (e.g., courier) number if abbreviations are more than one letter long. This will make the abbreviation definitions line up (e.g., specify `font = 2`, the default for courier). Ignored for `print` and `plot`.

`adj.to` If you didn't define `datadist` for all predictors, you will have to define adjustment settings for the undefined ones, e.g. `adj.to= list(age = 50, sex = "female")`.

`lp` Set to `FALSE` to suppress creation of an axis for scoring $X\beta$.

`lp.at` If `lp=TRUE`, `lp.at` may specify a vector of settings of $X\beta$. Default is to use `pretty(range of linear predictors, nint)`.

`fun` an optional function to transform the linear predictors, and to plot on another axis. If more than one transformation is plotted, put them in a list, e.g. `list(function(x) x/2, function(x) 2*x)`. Any function values equal to `NA` will be ignored.

`fun.at` function values to label on axis. Default `fun` evaluated at `lp.at`. If more than one `fun` was specified, using a vector for `fun.at` will cause all functions to be evaluated at the same argument values. To use different values, specify a list of vectors for `fun.at`, with elements corresponding to the different functions (lists of vectors also applies to `fun.lp.at` and `fun.side`).

`fun.lp.at` If you want to evaluate one of the functions at a different set of linear predictor values than may have been used in constructing the linear predictor axis, specify a vector or list of vectors of linear predictor values at which to evaluate the function. This is especially useful for discrete functions. The presence of this attribute also does away with the need for `nomogram` to compute numerical approximations of the inverse of the function. It also allows the user-supplied function to return `factor` objects, which is useful when e.g. a single tick mark position actually represents a range. If the `fun.lp.at` parameter is present, the `fun.at` vector for that function is ignored.

`funlabel` label for `fun` axis. If more than one function was given but funlabel is of length one, it will be duplicated as needed. If `fun` is a list of functions for which you specified names (see the final example below), these names will be used as labels.

`interact` When a continuous variable interacts with a discrete one, axes are constructed so that the continuous variable moves within the axis, and separate axes represent levels of interacting factors. For interactions between two continuous variables, all but the axis variable must have discrete levels defined in `interact`. For discrete interacting factors, you may specify levels to use in constructing the multiple axes. For continuous interacting factors, you must do this. Examples: `interact = list(age = seq(10,70,by=10), treat = c("A","B","D"))`.

`kint` for models such as the ordinal models with multiple intercepts, specifies which one to use in evaluating the linear predictor. Default is to use `fit$interceptRef` if it exists, or 1.

`conf.int` confidence levels to display for each scoring. Default is `FALSE` to display no confidence limits. Setting `conf.int` to `TRUE` is the same as setting it to `c(0.7, 0.9)`, with the line segment between the 0.7 and 0.9 levels shaded using gray scale.

`conf.lp` default is `"representative"` to group all linear predictors evaluated into deciles, and to show, for the linear predictor confidence intervals, only the mean linear predictor within the deciles along with the median standard error within the deciles. Set `conf.lp = "none"` to suppress confidence limits for the linear predictors, and to `"all"` to show all confidence limits.

`est.all` To plot axes for only the subset of variables named in `...`, set `est.all = FALSE`. Note: This option only works when zero has a special meaning for the variables that are omitted from the graph.

`posterior.summary` when operating on a Bayesian model such as a result of `blrm` specifies whether to use posterior mean (default) vs. posterior mode/median of parameter values in constructing the nomogram

`abbrev` Set to `TRUE` to use the `abbreviate` function to abbreviate levels of categorical factors, both for labeling tick marks and for axis titles. If you only want to abbreviate certain predictor variables, set `abbrev` to a vector of character strings containing their names.

`minlength` applies if `abbrev = TRUE`. Is the minimum abbreviation length passed to the `abbreviate` function. If you set `minlength = 1`, the letters of the alphabet are used to label tick marks for categorical predictors, and all letters are drawn no matter how close together they are. For labeling axes (interaction settings), `minlength = 1` causes `minlength = 4` to be used.

`maxscale` default maximum point score is 100

`nint` number of intervals to label for axes representing continuous variables. See `pretty`.

`vnames` By default, variable labels are used to label axes. Set `vnames = "names"` to instead use variable names.

`omit` vector of character strings containing names of variables for which to suppress drawing axes. Default is to show all variables.

verbose — set to TRUE to get printed output detailing how tick marks are chosen and labeled for function axes. This is useful in seeing how certain linear predictor values cannot be solved for using inverse linear interpolation on the (requested linear predictor values, function values at these lp values). When this happens you will see NAs in the verbose output, and the corresponding tick marks will not appear in the nomogram.

x — an object created by nomogram, or the x coordinate for a legend

dec — number of digits to the right of the decimal point, for rounding point scores in print.nomogram. Default is to round to the nearest whole number of points.

lplabel — label for linear predictor axis. Default is "Linear Predictor".

fun.side — a vector or list of vectors of side parameters for the axis function for labeling function values. Values may be 1 to position a tick mark label below the axis (the default), or 3 for above the axis. If for example an axis has 5 tick mark labels and the second and third will run into each other, specify fun.side=c(1,1,3,1,1) (assuming only one function is specified as fun).

col.conf — colors corresponding to conf.int.

conf.space — a 2-element vector with the vertical range within which to draw confidence bars, in units of 1=spacing between main bars. Four heights are used within this range (8 for the linear predictor if more than 16 unique values were evaluated), cycling them among separate confidence intervals to reduce overlapping.

label.every — Specify label.every = i to label on every ith tick mark.

force.label — set to TRUE to force every tick mark intended to be labeled to have a label plotted (whether the labels run into each other or not)

xfrac — fraction of horizontal plot to set aside for axis titles

cex.axis — character size for tick mark labels

cex.var — character size for axis titles (variable names)

col.grid — If left unspecified, no vertical reference lines are drawn. Specify a vector of length one (to use the same color for both minor and major reference lines) or two (corresponding to the color for the major and minor divisions, respectively) containing colors, to cause vertical reference lines to the top points scale to be drawn. For R, a good choice is col.grid = gray(c(0.8, 0.95)).

varname.label — In constructing axis titles for interactions, the default is to add (interacting.varname = level) on the right. Specify varname.label = FALSE to instead use "(level)".

varname.label.sep — If varname.label = TRUE, you can change the separator to something other than = by specifying this parameter.

ia.space — When multiple axes are draw for levels of interacting factors, the default is to group combinations related to a main effect. This is done by spacing the axes for the second to last of these within a group only 0.7 (by default) of the way down as compared with normal space of 1 unit.

tck — see tck under par

tcl — length of tick marks in nomogram

lmgp — spacing between numeric axis labels and axis (see par for mgp)

naxes — maximum number of axes to allow on one plot. If the nomogram requires more than one "page", the "Points" axis will be repeated at the top of each page when necessary.

points.label — a character string giving the axis label for the points scale

total.points.label — a character string giving the axis label for the total points scale

total.sep.page — set to TRUE to force the total points and later axes to be placed on a separate page

total.fun — a user-provided function that will be executed before the total points axis is drawn. Default is not to execute a function. This is useful e.g. when total.sep.page = TRUE and you wish to use locator to find the coordinates for positioning an abbreviation legend before it's too late and a new page is started (i.e., total.fun = function() print(locator(1))).

cap.labels — logical: should the factor labels have their first letter capitalized?

object — the result returned from nomogram

which — a character string giving the name of a variable for which to draw a legend with abbreviations of factor levels

y — y-coordinate to pass to the legend function. This is the upper left corner of the legend box. You can omit y if x is a list with named elements x and y. To use the mouse to locate the legend, specify locator(1) for x. For print, x is the result of nomogram.

ncol — the number of columns to form in drawing the legend.

Details

A variable is considered to be discrete if it is categorical or ordered or if datadist stored values for it (meaning it had <11 unique values). A variable is said to be indirectly related to another variable if the two are related by some interaction. For example, if a model has variables a, b, c, d, and the interactions are a:c and c:d, variable d is indirectly related to variable a. The complete list of variables related to a is c, d. If an axis is made for variable a, several axes will actually be drawn, one for each combination of c and d specified in interact.

Note that with a caliper, it is easy to continually add point scores for individual predictors, and then to place the caliper on the upper "Points" axis (with extrapolation if needed). Then transfer these points to the "Total Points" axis. In this way, points can be added without writing them down.

Confidence limits for an individual predictor score are really confidence limits for the entire linear predictor, with other predictors set to adjustment values. If lp = TRUE, all confidence bars for all linear predictor values evaluated are drawn. The extent to which multiple confidence bars of differing widths appear at the same linear predictor value means that precision depended on how the linear predictor was arrived at (e.g., a certain value may be realized from a setting of a certain predictor that was associated with a large standard error on the regression coefficients for that predictor).

On occasion, you may want to reverse the regression coefficients of a model to make the "points" scales reverse direction. For parametric survival models, which are stated in terms of increasing regression effects meaning longer survival (the opposite of a Cox model), just do something like fit$coefficients <- -fit$coefficients before invoking nomogram, and if you add function axes, negate the function arguments. For the Cox model, you also need to negate fit$center. If you omit lp.at, also negate fit$linear.predictors.

Value

a list of class "nomogram" that contains information used in plotting the axes. If you specified abbrev = TRUE, a list called abbrev is also returned that gives the abbreviations used for tick mark labels, if any. This list is useful for making legends and is used by legend.nomabbrev (see the last example). The returned list also has components called total.points, lp, and the function axis names. These components have components x (at argument vector given to axis), y (pos for axis), and x.real, the x-coordinates appearing on tick mark labels. An often useful result is stored in the list of data for each axis variable, namely the exact number of points that correspond to each tick mark on that variable's axis.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
<fh@fharrell.com>

References

Banks J: Nomograms. Encylopedia of Statistical Sciences, Vol 6. Editors: S Kotz and NL Johnson. New York: Wiley; 1985.

Lubsen J, Pool J, van der Does, E: A practical device for the application of a diagnostic or prognostic function. Meth. Inform. Med. 17:127–129; 1978.

Wikipedia: Nomogram, https://en.wikipedia.org/wiki/Nomogram.

See Also

rms, plot.Predict, ggplot.Predict, plot.summary.rms, axis, pretty, approx, latexrms, rmsMisc

Examples

```
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
d <- data.frame(age = rnorm(n, 50, 10),
                blood.pressure = rnorm(n, 120, 15),
                cholesterol = rnorm(n, 200, 25),
                sex = factor(sample(c('female','male'), n,TRUE)))

# Specify population model for log odds that Y=1
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
d <- upData(d,
  L = .4*(sex=='male') + .045*(age-50) +
       (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male')),
  y = ifelse(runif(n) < plogis(L), 1, 0))

ddist <- datadist(d); options(datadist='ddist')


f <- lrm(y ~ lsp(age,50) + sex * rcs(cholesterol, 4) + blood.pressure,
```

```
         data=d)
nom <- nomogram(f, fun=function(x)1/(1+exp(-x)),  # or fun=plogis
    fun.at=c(.001,.01,.05,seq(.1,.9,by=.1),.95,.99,.999),
    funlabel="Risk of Death")
#Instead of fun.at, could have specified fun.lp.at=logit of
#sequence above - faster and slightly more accurate
plot(nom, xfrac=.45)
print(nom)
nom <- nomogram(f, age=seq(10,90,by=10))
plot(nom, xfrac=.45)
g <- lrm(y ~ sex + rcs(age, 3) * rcs(cholesterol, 3), data=d)
nom <- nomogram(g, interact=list(age=c(20,40,60)),
                conf.int=c(.7,.9,.95))
plot(nom, col.conf=c(1,.5,.2), naxes=7)

require(survival)
w <- upData(d,
            cens = 15 * runif(n),
            h = .02 * exp(.04 * (age - 50) + .8 * (sex == 'Female')),
            d.time = -log(runif(n)) / h,
            death = ifelse(d.time <= cens, 1, 0),
            d.time = pmin(d.time, cens))


f <- psm(Surv(d.time,death) ~ sex * age, data=w, dist='lognormal')
med  <- Quantile(f)
surv <- Survival(f)  # This would also work if f was from cph
plot(nomogram(f, fun=function(x) med(lp=x), funlabel="Median Survival Time"))
nom <- nomogram(f, fun=list(function(x) surv(3, x),
                            function(x) surv(6, x)),
            funlabel=c("3-Month Survival Probability",
                       "6-month Survival Probability"))
plot(nom, xfrac=.7)

## Not run:
nom <- nomogram(fit.with.categorical.predictors, abbrev=TRUE, minlength=1)
nom$x1$points   # print points assigned to each level of x1 for its axis
#Add legend for abbreviations for category levels
abb <- attr(nom, 'info')$abbrev$treatment
legend(locator(1), abb$full, pch=paste(abb$abbrev,collapse=''),
       ncol=2, bty='n')  # this only works for 1-letter abbreviations
#Or use the legend.nomabbrev function:
legend.nomabbrev(nom, 'treatment', locator(1), ncol=2, bty='n')

## End(Not run)


#Make a nomogram with axes predicting probabilities Y>=j for all j=1-3
#in an ordinal logistic model, where Y=0,1,2,3
w <- upData(w, Y = ifelse(y==0, 0, sample(1:3, length(y), TRUE)))
g <- lrm(Y ~ age+rcs(cholesterol,4) * sex, data=w)
fun2 <- function(x) plogis(x-g$coef[1]+g$coef[2])
fun3 <- function(x) plogis(x-g$coef[1]+g$coef[3])
```

```
f <- Newlabels(g, c(age='Age in Years'))
#see Design.Misc, which also has Newlevels to change
#labels for levels of categorical variables
g <- nomogram(f, fun=list('Prob Y>=1'=plogis, 'Prob Y>=2'=fun2,
                      'Prob Y=3'=fun3),
         fun.at=c(.01,.05,seq(.1,.9,by=.1),.95,.99))
plot(g, lmgp=.2, cex.axis=.6)
options(datadist=NULL)
```

---

npsurv *Nonparametric Survival Estimates for Censored Data*

Description

Computes an estimate of a survival curve for censored data using either the Kaplan-Meier or the Fleming-Harrington method or computes the predicted survivor function. For competing risks data it computes the cumulative incidence curve. This calls the `survival` package's `survfit.formula` function. Attributes of the event time variable are saved (label and units of measurement).

For competing risks the second argument for `Surv` should be the event state variable, and it should be a factor variable with the first factor level denoting right-censored observations.

Usage

```
npsurv(formula, data=environment(formula),
               subset, weights, na.action=na.delete, ...)
```

Arguments

`formula` a formula object, which must have a `Surv` object as the response on the left of the ~ operator and, if desired, terms separated by + operators on the right. One of the terms may be a `strata` object. For a single survival curve the right hand side should be ~ 1.

`data`, `subset`, `weights`, `na.action` see `survfit.formula`

`...` see `survfit.formula`

Details

see `survfit.formula` for details

Value

an object of class "`npsurv`" and "`survfit`". See `survfit.object` for details. Methods defined for `survfit` objects are `print`, `summary`, `plot`,`lines`, and `points`.

Author(s)

Thomas Lumley <`tlumley@u.washington.edu`> and Terry Therneau

See Also

survfit.cph for survival curves from Cox models. print, plot, lines, coxph, strata, survplot, ggplot.npsurv

Examples

```
require(survival)
# fit a Kaplan-Meier and plot it
fit <- npsurv(Surv(time, status) ~ x, data = aml)
plot(fit, lty = 2:3)
legend(100, .8, c("Maintained", "Nonmaintained"), lty = 2:3)
ggplot(fit)   # prettier than plot()

# Here is the data set from Turnbull
#  There are no interval censored subjects, only left-censored (status=3),
#  right-censored (status 0) and observed events (status 1)
#
#                              Time
#                         1    2   3   4
# Type of observation
#           death        12    6   2   3
#          losses         3    2   0   3
#      late entry         2    4   2   5
#
tdata <- data.frame(time   = c(1,1,1,2,2,2,3,3,3,4,4,4),
                    status = rep(c(1,0,2),4),
                    n      = c(12,3,2,6,2,4,2,0,2,3,3,5))
fit  <- npsurv(Surv(time, time, status, type='interval') ~ 1,
               data=tdata, weights=n)

#
# Time to progression/death for patients with monoclonal gammopathy
# Competing risk curves (cumulative incidence)
# status variable must be a factor with first level denoting right censoring
m <- upData(mgus1, stop = stop / 365.25, units=c(stop='years'),
            labels=c(stop='Follow-up Time'), subset=start == 0)
f <- npsurv(Surv(stop, event) ~ 1, data=m)

# CI curves are always plotted from 0 upwards, rather than 1 down
plot(f, fun='event', xmax=20, mark.time=FALSE,
     col=2:3, xlab="Years post diagnosis of MGUS")
text(10, .4, "Competing Risk: death", col=3)
text(16, .15,"Competing Risk: progression", col=2)

# Use survplot for enhanced displays of cumulative incidence curves for
# competing risks

survplot(f, state='pcm', n.risk=TRUE, xlim=c(0, 20), ylim=c(0, .5), col=2)
survplot(f, state='death', add=TRUE, col=3)

f <- npsurv(Surv(stop, event) ~ sex, data=m)
survplot(f, state='death', n.risk=TRUE, conf='diffbands')
```

Ocens — *Censored Ordinal Variable*

Description

Combines two variables `a, b` into a 2-column matrix, preserving `label` and `units` attributes and converting character or factor variables into integers and added a `levels` attribute. This is used to combine censoring points with regular points. If both variables are already factors, their levels are distinctly combined starting with the levels for `a`. Character variables are converted to factors.

Usage

```
Ocens(a, b = a)
```

Arguments

`a`	variable for first column
`b`	variable for second column

Details

Left censored values will have `-Inf` for `a` and right-censored values will have `Inf` for `b`. Interval-censored observations will have `b > a` and both finite. For factor or character variables it only makes sense to have interval censoring.

If there is no censoring, `a` is returned as an ordinary vector, with `label` and `units` attributes.

Value

a numeric matrix of class `Ocens`

Author(s)

Frank Harrell

Ocens2ord — *Recode Censored Ordinal Variable*

Description

Creates a 2-column integer matrix that handles left- right- and interval-censored ordinal or continuous values for use in [rmsb::blrm()] and [orm()]. A pair of values ‘[a, b]‘ represents an interval-censored value known to be in the interval ‘[a, b]‘ inclusive of ‘a‘ and ‘b‘. Left censored values are coded as ‘(-Infinity, b)‘ and right-censored as ‘(a, Infinity)‘, both of these intervals being open at the finite endpoints. Open left and right censoring intervals are created by adding a small increment (subtracting for left censoring) to ‘a‘ or ‘b‘. When this occurs at the outer limits, new ordinal categories will be created by ‘orm‘ to capture the real and unique information in outer censored values. For example if the highest uncensored value is 10 and there is a right-censored value in the data at 10, a new category ‘10+‘ is created, separate from the category for ‘10‘. So it is assumed that if an exact value of 10 was observed, the pair of values for that observation would not be coded as ‘(10, Infinity)‘.

Usage

```
Ocens2ord(
  y,
  precision = 7,
  maxit = 10,
  nponly = FALSE,
  cons = c("intervals", "data", "none"),
  verbose = FALSE
)
```

Arguments

`y`	an ‘Ocens‘ object, which is a 2-column numeric matrix, or a regular vector representing a ‘factor‘, numeric, integer, or alphabetically ordered character strings. Censoring points have values of ‘Inf‘ or ‘-Inf‘.
`precision`	when ‘y‘ columns are numeric, values may need to be rounded to avoid unpredictable behavior with `unique()` with floating-point numbers. Default is to 7 decimal places. See [this](https://hbiostat.org/r/rms/unique-float/) for more details.
`maxit`	maximum number of iterations allowed in the interval consolidation process when ‘cons='data'‘
`nponly`	set to ‘TRUE‘ to return a list containing the survival curve estimates before interval consolidation, using [icenReg::ic_np()]
`cons`	set to ‘'none'‘ to not consolidate intervals when the survival estimate stays constant; this will likely cause a lot of trouble with zero cell probabilities during maximum likelihood estimation. The default is to consolidate consecutive intervals. Set ‘cons='data'‘ to change the raw data values to make observed intervals wider, in an iterative manner until no more consecutive tied survival estimates remain.
`verbose`	set to ‘TRUE‘ to print information messages. Set ‘verbose‘ to a number greater than 1 to get more information printed, such as the estimated survival curve at each stage of consolidation.

Details

The intervals that drive the coding of the input data into numeric ordinal levels are the Turnbull intervals computed by the non-exported ‘findMaximalIntersections‘ function in the ‘icenReg‘ package, which handles all three types of censoring. These are defined in the ‘levels‘ and ‘upper‘ attributes of the object returned by ‘Ocens‘. Sometimes consecutive Turnbull intervals contain the same statistical information likelihood function-wise, leading to the same survival estimates over two ore more consecutive intervals. This leads to zero probabilities of involved ordinal values, preventing ‘orm‘ from computing a valid log-likeliihood. A limited about of interval consolidation is done by ‘Ocens‘ to alleviate this problem. Depending on the value of ‘cons‘ this consolidation is done by intervals (preferred) or by changing the raw data. If ‘verbose=TRUE‘, information about the actions taken is printed.

When both input variables are ‘factor‘s it is assumed that the one with the higher number of levels is the one that correctly specifies the order of levels, and that the other variable does not contain any additional levels. If the variables are not ‘factor‘s it is assumed their original values provide the orderings. A left-censored point is is coded as having ‘-Inf‘ as a lower limit, and a right-censored point is coded as having ‘Inf‘ as an upper limit. As with most censored-data methods, modeling functions assumes that censoring is independent of the response variable values that would have been measured had censoring not occurred. ‘Ocens‘ creates a 2-column integer matrix suitable for ordinal regression. Attributes of the returned object give more information.

Value

a 2-column integer matrix of class ‘"Ocens"‘ with an attribute ‘levels‘ (ordered), and if there are zero-width intervals arising from censoring, an attribute ‘upper‘ with the vector of upper limits. Left-censored values are coded as ‘-Inf‘ in the first column of the returned matrix, and right-censored values as ‘Inf‘. When the original variables were ‘factor‘s, these are factor levels, otherwise are numerically or alphabetically sorted distinct (over ‘a‘ and ‘b‘ combined) values. When the variables are not factors and are numeric, other attributes ‘median‘, ‘range‘, ‘label‘, and ‘npsurv‘ are also returned. ‘median‘ is the median of the uncensored values on the origiinal scale. ‘ranges‘ is a 3-element list, each element a 2-vector range. The element named ‘y‘ is the range of original data values before adjustments. The ‘u‘ element is a 2-vector range of uncensored values before adjustment, and the ‘c‘ element contains the lowest left censoring point and highest right-censored point. Getting back to the main returned variables, ‘label‘ is the ‘label‘ attribute from the first of ‘a, b‘ having a label. ‘npsurv‘ is the estimated survival curve (with elements ‘time‘ and ‘surv‘) from the ‘icenReg‘ package after any interval consolidation. If the argument ‘npsurv=TRUE‘ was given, this ‘npsurv‘ list before consolidation is returned and no other calculations are done. When the variables are factor or character, the median of the integer versions of variables for uncensored observations is returned as attribute ‘mid‘. A final attribute ‘freq‘ is the vector of frequencies of occurrences of all values. ‘freq‘ aligns with ‘levels‘. A ‘units‘ attribute is also included. Finally there are two 3-vectors ‘Ncens1‘ and ‘Ncens2‘, the first containing the original number of left, right, and interval-censored observations and the second containing the frequencies after altering some of the data. For example, observations that are right-censored at or beyond the highest uncensored value are coded as uncensored to get the correct likelihood component in ‘orm.fit‘. When only right censoring is present and there are censored observations at or beyond the highest uncensored point, another attribute ‘rt_cens_beyond‘ is included in the returned list. It has elements ‘newlevel‘ which is the numeric uncensored value assigned to these observations, and ‘range‘ which is a 2-vector containing the lowest and highest censored values beyond the last uncensored value.

Author(s)

Frank Harrell

`Ocens2Surv` *Ocens2Surv*

Description

Converts an `Ocens` object to the simplest `Surv` object that works for the types of censoring that are present in the data.

Usage

```
Ocens2Surv(Y)
```

Arguments

`Y`	an `Ocens` object

Value

a `Surv` object

Examples

```
Y <- Ocens(1:3, c(1, Inf, 3))
Ocens2Surv(Y)
```

`Olinks` *Likehood-Based Statistics for Other Links for orm Fits*

Description

Likehood-Based Statistics for Other Links for orm Fits

Usage

```
Olinks(
  object,
  links = c("logistic", "probit", "loglog", "cloglog"),
  dec = 3,
  gradtol = 0.001
)
```

Arguments

object	an object created by orm with x=TRUE, y=TRUE
links	a vector of links to consider other than the one used to get object
dec	number of digits to the right of the decimal place to round statistics to
gradtol	tolerance for convergence on the absolute gradient; see lrm.fit and orm.fit.

Value

data frame. The R2 column is from the last adjusted R^2 computed by orm, which adjustes for the effective sample size and the number of betas.

Author(s)

Frank Harrell

Examples

```
## Not run:
f <- orm(y ~ x1 + x2, family='loglog', x=TRUE, y=TRUE)
Olinks(f)

## End(Not run)
```

---

ols — *Linear Model Estimation Using Ordinary Least Squares*

Description

Fits the usual weighted or unweighted linear regression model using the same fitting routines used by lm, but also storing the variance-covariance matrix var and using traditional dummy-variable coding for categorical factors. Also fits unweighted models using penalized least squares, with the same penalization options as in the lrm function. For penalized estimation, there is a fitter function call lm.pfit.

Usage

```
ols(formula, data=environment(formula), weights, subset, na.action=na.delete,
    method="qr", model=FALSE,
    x=FALSE, y=FALSE, se.fit=FALSE, linear.predictors=TRUE,
    penalty=0, penalty.matrix, tol=.Machine$double.eps, sigma,
    var.penalty=c('simple','sandwich'), ...)
```

Arguments

`formula` an S formula object, e.g.
Y ~ rcs(x1,5)*lsp(x2,c(10,20))

`data` name of an S data frame containing all needed variables. Omit this to use a data frame already in the S "search list".

`weights` an optional vector of weights to be used in the fitting process. If specified, weighted least squares is used with weights `weights` (that is, minimizing $sum(w*e^2)$); otherwise ordinary least squares is used.

`subset` an expression defining a subset of the observations to use in the fit. The default is to use all observations. Specify for example `age>50 & sex="male"` or `c(1:100,200:300)` respectively to use the observations satisfying a logical expression or those having row numbers in the given vector.

`na.action` specifies an S function to handle missing data. The default is the function `na.delete`, which causes observations with any variable missing to be deleted. The main difference between `na.delete` and the S-supplied function `na.omit` is that `na.delete` makes a list of the number of observations that are missing on each variable in the model. The `na.action` is usally specified by e.g. `options(na.action="na.delete")`.

`method` specifies a particular fitting method, or `"model.frame"` instead to return the model frame of the predictor and response variables satisfying any subset or missing value checks.

`model` default is `FALSE`. Set to `TRUE` to return the model frame as element `model` of the fit object.

`x` default is `FALSE`. Set to `TRUE` to return the expanded design matrix as element `x` (without intercept indicators) of the returned fit object. Set both `x=TRUE` if you are going to use the `residuals` function later to return anything other than ordinary residuals.

`y` default is `FALSE`. Set to `TRUE` to return the vector of response values as element `y` of the fit.

`se.fit` default is `FALSE`. Set to `TRUE` to compute the estimated standard errors of the estimate of $X\beta$ and store them in element `se.fit` of the fit.

`linear.predictors`
set to `FALSE` to cause predicted values not to be stored

`penalty` see `lrm`

`penalty.matrix` see `lrm`

`tol` tolerance for information matrix singularity

`sigma` If `sigma` is given, it is taken as the actual root mean squared error parameter for the model. Otherwise `sigma` is estimated from the data using the usual formulas (except for penalized models). It is often convenient to specify `sigma=1` for models with no error, when using `fastbw` to find an approximate model that predicts predicted values from the full model with a given accuracy.

`var.penalty` the type of variance-covariance matrix to be stored in the `var` component of the fit when penalization is used. The default is the inverse of the penalized

information matrix. Specify `var.penalty="sandwich"` to use the sandwich estimator (see below under `var`), which limited simulation studies have shown yields variances estimates that are too low.

`...` arguments to pass to `lm.wfit` or `lm.fit`

Details

For penalized estimation, the penalty factor on the log likelihood is $-0.5\beta' P\beta/\sigma^2$, where P is defined above. The penalized maximum likelihood estimate (penalized least squares or ridge estimate) of β is $(X'X+P)^{-1}X'Y$. The maximum likelihood estimate of σ^2 is $(sse+\beta' P\beta)/n$, where `sse` is the sum of squared errors (residuals). The `effective.df.diagonal` vector is the diagonal of the matrix $X'X/(sse/n)\sigma^2(X'X+P)^{-1}$.

Value

the same objects returned from `lm` (unless `penalty` or `penalty.matrix` are given - then an abbreviated list is returned since `lm.pfit` is used as a fitter) plus the design attributes (see `rms`). Predicted values are always returned, in the element `linear.predictors`. The vectors or matrix stored if `y=TRUE` or `x=TRUE` have rows deleted according to `subset` and to missing data, and have names or row names that come from the data frame used as input data. If `penalty` or `penalty.matrix` is given, the `var` matrix returned is an improved variance-covariance matrix for the penalized regression coefficient estimates. If `var.penalty="sandwich"` (not the default, as limited simulation studies have found it provides variance estimates that are too low) it is defined as $\sigma^2(X'X+P)^{-1}X'X(X'X+P)^{-1}$, where P is `penalty factors * penalty.matrix`, with a column and row of zeros added for the intercept. When `var.penalty="simple"` (the default), `var` is $\sigma^2(X'X+P)^{-1}$. The returned list has a vector `stats` with named elements `n`, `Model L.R.`, `d.f.`, `R2`, `g`, `Sigma`. `Model L.R.` is the model likelihood ratio χ^2 statistic, and `R2` is R^2. For penalized estimation, `d.f.` is the effective degrees of freedom, which is the sum of the elements of another vector returned, `effective.df.diagonal`, minus one for the intercept. `g` is the g-index. `Sigma` is the penalized maximum likelihood estimate (see below).

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

`rms`, `rms.trans`, `anova.rms`, `summary.rms`, `predict.rms`, `fastbw`, `validate`, `calibrate`, `Predict`, `specs.rms`, `cph`, `lrm`, `which.influence`, `lm`, `summary.lm`, `print.ols`, `residuals.ols`, `latex.ols`, `na.delete`, `na.detail.response`, `datadist`, `pentrace`, `vif`, `abs.error.pred`

Examples

```
set.seed(1)
x1 <- runif(200)
x2 <- sample(0:3, 200, TRUE)
distance <- (x1 + x2/3 + rnorm(200))^2
d <- datadist(x1,x2)
```

```
options(datadist="d")   # No d -> no summary, plot without giving all details


f <- ols(sqrt(distance) ~ rcs(x1,4) + scored(x2), x=TRUE)
# could use d <- datadist(f); options(datadist="d") at this point,
# but predictor summaries would not be stored in the fit object for
# use with Predict, summary.rms.  In that case, the original
# dataset or d would need to be accessed later, or all variable values
# would have to be specified to summary, plot
anova(f)
which.influence(f)
summary(f)
summary.lm(f)    # will only work if penalty and penalty.matrix not used


# Fit a complex model and approximate it with a simple one
x1 <- runif(200)
x2 <- runif(200)
x3 <- runif(200)
x4 <- runif(200)
y <- x1 + x2 + rnorm(200)
f    <- ols(y ~ rcs(x1,4) + x2 + x3 + x4)
pred <- fitted(f)   # or predict(f) or f$linear.predictors
f2   <- ols(pred ~ rcs(x1,4) + x2 + x3 + x4, sigma=1)
# sigma=1 prevents numerical problems resulting from R2=1
fastbw(f2, aics=100000)
# This will find the best 1-variable model, best 2-variable model, etc.
# in predicting the predicted values from the original model
options(datadist=NULL)
```

---

ordESS *ordESS*

Description

Ordinal Model Effective Sample Size

Usage

```
ordESS(fit)
```

Arguments

`fit`	a model fitted by `orm` with `y=TRUE, lpe=TRUE`

Details

For a standard ordinal model fitted with `orm`, returns the effective sample size (ESS) component of the `stats` part of the fit object if there were no censored data. Otherwise `ordESS` assumes that `y=TRUE` and `lpe=TRUE` were given to `orm`, and an analysis of the effective sample size per censored

observation is given, as a function of the censoring time, or in the case of interval censored data, o function of the width of the interval.

Value

a ggplot2 object

Author(s)

Frank Harrell

Examples

```
## Not run:
f <- orm(Ocens(y1, y2) ~ x, y=TRUE, lpe=TRUE)
ordESS(f)

## End(Not run)
```

---

ordParallel *Check Parallelism Assumption of Ordinal Semiparametric Models*

Description

orm models are refitted as a series of binary models for a sequence of cutoffs on the dependent variable. Regression coefficients from this sequence are plotted against cutoffs using ggplot2 with one panel per regression coefficient. When censoring is present, whether or not Y is greater than or equal to the current cutoff is not always possible, and such observations are ignored.

Usage

```
ordParallel(
  fit,
  which,
  terms = onlydata,
  m,
  maxcuts = 75,
  lp = FALSE,
  onlydata = FALSE,
  scale = c("iqr", "none"),
  conf.int = 0.95,
  alpha = 0.15
)
```

Arguments

`fit`	a fit object from `orm` with `x=TRUE, y=TRUE` in effect
`which`	specifies which columns of the design matrix are assessed. By default, all columns are analyzed.
`terms`	set to `TRUE` to collapse all components of each predictor into a single column weighted by the original regression coefficients but scaled according to `scale`. This means that each predictor will have a regression coefficient of 1.0 when refitting the original model on this transformed X matrix, before any further scaling. Plots will then show the relative effects over time, i.e., the slope of these combined columns over cuts on Y, so that deviations indicate non-parallelism. But since in this case only relative effects are shown, a weak predictor may be interpreted as having an exagerrated y-dependency if `scale='none'`. `terms` detauls to `TRUE` when `onlydata=TRUE`.
`m`	the lowest cutoff is chosen as the first Y value having at meast `m` observations to its left, and the highest cutoff is chosen so that there are at least `m` observations tot he right of it. Cutoffs are equally spaced between these values. If omitted, `m` is set to the minimum of 50 and one quarter of the sample size.
`maxcuts`	the maximum number of cutoffs analyzed
`lp`	plot the effect of the linear predictor across cutpoints instead of analyzing individual predictors
`onlydata`	set to `TRUE` to return a data frame suitable for modeling effects of cuts, instead of constructing a graph. The returned data frame has variables `Ycut, Yge_cut, obs`, and the original names of the predictors. `Ycut` has the cutpoint on the original scale. `Yge_cut` is `TRUE/FALSE` dependent on whether the Y variable is greater than or equal to `Ycut`, with `NA` if censoring prevented this determination. The `obs` variable is useful for passing as the `cluster` argument to `robcov()` to account for the high correlations in regression coefficients across cuts. See the example which computes Wald tests for parallelism where the `Ycut` dependence involves a spline function. But since `terms` was used, each predictor is reduced to a single degree of freedom.
`scale`	applies to `terms=TRUE`; set to `'none'` to leave the predictor terms scaled by regression coefficient so the coefficient of each term in the overall fit is 1.0. The default is to scale terms by the interquartile-range (Gini's mean difference if IQR is zero) of the term. This prevents changes in weak predictors over different cutoffs from being impressive.
`conf.int`	confidence level for computing Wald confidence intervals for regression coefficients. Set to 0 to suppress confidence bands.
`alpha`	saturation for confidence bands

Details

Whenver a cut gives rise to extremely high standard error for a regression coefficient, the confidence limits are set to `NA`. Unreasonable standard errors are determined from the confidence interval width exceeding 7 times the standard error at the middle Y cut.

Value

ggplot2 object or a data frame

Author(s)

Frank Harrell

Examples

```
## Not run:
f <- orm(..., x=TRUE, y=TRUE)
ordParallel(f, which=1:5)  # first 5 betas

getHdata(nhgh)
set.seed(1)
nhgh$ran <- runif(nrow(nhgh))
f <- orm(gh ~ rcs(age, 4) + ran, data=nhgh, x=TRUE, y=TRUE)
ordParallel(f)  # one panel per parameter (multiple parameters per predictor)
dd <- datadist(nhgh); options(datadist='dd')
ordParallel(f, terms=TRUE)
d <- ordParallel(f, maxcuts=30, onlydata=TRUE)
dd2 <- datadist(d); options(datadist='dd2')  # needed for plotting
g <- orm(Yge_cut ~ (age + ran) * rcs(Ycut, 4), data=d, x=TRUE, y=TRUE)
h <- robcov(g, d$obs)
anova(h)
qu <- quantile(d$age, c(1, 3)/4)
qu
cuts <- sort(unique(d$Ycut))
cuts
z <- contrast(h, list(age=qu[2], Ycut=cuts),
                 list(age=qu[1], Ycut=cuts))
z <- as.data.frame(z[.q(Ycut, Contrast, Lower, Upper)])
ggplot(z, aes(x=Ycut, y=Contrast)) + geom_line() +
  geom_ribbon(aes(ymin=Lower, ymax=Upper), alpha=0.2)

## End(Not run)
```

---

orm *Ordinal Regression Model*

Description

Fits ordinal cumulative probability models for continuous or ordinal response variables, efficiently allowing for a large number of intercepts by capitalizing on the information matrix being sparse. Five different distribution functions are implemented, with the default being the logistic (i.e., the proportional odds model). The ordinal cumulative probability models are stated in terms of exceedance probabilities ($Prob[Y \geq y|X]$) so that as with OLS larger predicted values are associated with larger Y. This is important to note for the asymmetric distributions given by the log-log and complementary log-log families, for which negating the linear predictor does not result in

$Prob[Y < y|X]$. The `family` argument is defined in `orm.fit`. The model assumes that the inverse of the assumed cumulative distribution function, when applied to one minus the true cumulative distribution function and plotted on the y-axis (with the original y on the x-axis) yields parallel curves (though not necessarily linear). This can be checked by plotting the inverse cumulative probability function of one minus the empirical distribution function, stratified by `X`, and assessing parallelism. Note that parametric regression models make the much stronger assumption of linearity of such inverse functions.

For the `print` method, format of output is controlled by the user previously running `options(prType="lang")` where `lang` is `"plain"` (the default), `"latex"`, or `"html"`. When using html with Quarto or RMarkdown, `results='asis'` need not be written in the chunk header.

`Quantile.orm` creates an R function that computes an estimate of a given quantile for a given value of the linear predictor (which was assumed to use thefirst intercept). It uses a linear interpolation method by default, but you can override that to use a discrete method by specifying `method="discrete"` when calling the function generated by `Quantile`. Optionally a normal approximation for a confidence interval for quantiles will be computed using the delta method, if `conf.int > 0` is specified to the function generated from calling `Quantile` and you specify `X`. In that case, a `"lims"` attribute is included in the result computed by the derived quantile function.

Usage

```
orm(formula, data=environment(formula),
    subset, na.action=na.delete, method="orm.fit",
    family=c("logistic", "probit", "loglog", "cloglog", "cauchit"),
    model=FALSE, x=FALSE, y=FALSE, lpe=FALSE,
    linear.predictors=TRUE, se.fit=FALSE,
    penalty=0, penalty.matrix,
    var.penalty=c('simple','sandwich'), scale=FALSE,
    maxit=30, weights, normwt=FALSE, ...)

## S3 method for class 'orm'
print(x, digits=4, r2=c(0,2,4), coefs=TRUE, pg=FALSE,
    intercepts=x$non.slopes < 10, title, ...)

## S3 method for class 'orm'
Quantile(object, codes=FALSE, ...)
```

Arguments

`formula`	a formula object. An `offset` term can be included. The offset causes fitting of a model such as $logit(Y = 1) = X\beta + W$, where W is the offset variable having no estimated coefficient. The response variable can be any data type; `orm` converts it in alphabetic or numeric order to a factor variable and recodes it 1,2,... internally.
`data`	data frame to use. Default is the current frame.
`subset`	logical expression or vector of subscripts defining a subset of observations to analyze
`na.action`	function to handle `NA`s in the data. Default is `na.delete`, which deletes any observation having response or predictor missing, while preserving the attributes

of the predictors and maintaining frequencies of deletions due to each variable in the model. This is usually specified using `options(na.action="na.delete")`.

`method` name of fitting function. Only allowable choice at present is `orm.fit`.

`family` character value specifying the distribution family, which is one of the following: `"logistic"`, `"probit"`, `"loglog"`, `"cloglog"`, `"cauchit"`, corresponding to logistic (the default), Gaussian, Cauchy, Gumbel maximum ($exp(-exp(-x))$; extreme value type I), and Gumbel minimum ($1 - exp(-exp(x))$) distributions. These are the cumulative distribution functions assumed for $Prob[Y \geq y|X]$. The default is `"logistic"`.

`model` causes the model frame to be returned in the fit object

`x` causes the expanded design matrix (with missings excluded) to be returned under the name `x`. For `print`, an object created by `orm`.

`y` causes the response variable (with missings excluded) to be returned under the name `y`.

`lpe` set `lpe=TRUE` to store the vector of likelihood probability elements in the fit object in a list element named `lpe`. This will enable the `ordESS` function to summarize the effective sample sizes of any censored observations.

`linear.predictors`
causes the predicted X beta (with missings excluded) to be returned under the name `linear.predictors`. The first intercept is used.

`se.fit` causes the standard errors of the fitted values (on the linear predictor scale) to be returned under the name `se.fit`. The middle intercept is used.

`penalty` see `lrm`

`penalty.matrix` see `lrm`

`var.penalty` see `lrm`

`scale` set to `TRUE` to subtract column means and divide by column standard deviations of the design matrix before fitting, and to back-solve for the un-normalized covariance matrix and regression coefficients. This can sometimes make the model converge for very large sample sizes where for example spline or polynomial component variables create scaling problems leading to loss of precision when accumulating sums of squares and crossproducts.

`maxit` maximum number of iterations to allow in `orm.fit`

`weights` a vector (same length as `y`) of possibly fractional case weights

`normwt` set to `TRUE` to scale `weights` so they sum to the length of `y`; useful for sample surveys as opposed to the default of frequency weighting

`...` arguments that are passed to `orm.fit`, or from `print`, to `prModFit`. Ignored for `Quantile`. One of the most important arguments is `family`.

`digits` number of significant digits to use

`r2` vector of integers specifying which R^2 measures to print, with 0 for Nagelkerke R^2 and 1:4 corresponding to the 4 measures computed by `R2Measures`. Default is to print Nagelkerke (labeled R2) and second and fourth `R2Measures` which are the measures adjusted for the number of predictors, first for the raw sample size then for the effective sample size, which here is from the formula for the approximate variance of a log odds ratio in a proportional odds model.

pg	set to TRUE to print g-indexes
coefs	specify coefs=FALSE to suppress printing the table of model coefficients, standard errors, etc. Specify coefs=n to print only the first n regression coefficients in the model.
intercepts	By default, intercepts are only printed if there are fewer than 10 of them. Otherwise this is controlled by specifying intercepts=FALSE or TRUE.
title	a character string title to be passed to prModFit. Default is constructed from the name of the distribution family.
object	an object created by orm
codes	if TRUE, uses the integer codes $1, 2, \ldots, k$ for the k-level response in computing the predicted quantile

Value

The returned fit object of orm contains the following components in addition to the ones mentioned under the optional arguments.

call	calling expression
freq	table of frequencies for Y in order of increasing Y
stats	vector with the following elements: number of observations used in the fit, effective sample size ESS, number of unique Y values, median Y from among the observations used int he fit, maximum absolute value of first derivative of log likelihood, model likelihood ratio χ^2, d.f., P-value, score χ^2 statistic (if no initial values given), P-value, Spearman's ρ rank correlation between the linear predictor and Y, the Nagelkerke R^2 index, R^2 indexes computed by R2Measures, the g-index, gr (the g-index on the odds ratio scale), and pdm (the mean absolute difference between 0.5 and the predicted probability that $Y \geq$ the marginal median). In the case of penalized estimation, the "Model L.R." is computed without the penalty factor, and "d.f." is the effective d.f. from Gray's (1992) Equation 2.9. The P-value uses this corrected model L.R. χ^2 and corrected d.f. The score chi-square statistic uses first derivatives which contain penalty components.
fail	set to TRUE if convergence failed (and maxiter>1) or if a singular information matrix is encountered
coefficients	estimated parameters
var	estimated variance-covariance matrix (inverse of information matrix) for the middle intercept and regression coefficients. See lrm for details if penalization is used.
effective.df.diagonal	see lrm
family	the character string for family.
famfunctions	a vector of expressions containing functions for the cumulative probability, inverse cumulative probability, derivative, second derivative, and derivative as a function of only x

trans	a list of functions for the choice of family, with elements cumprob (the cumulative probability distribution function), inverse (inverse of cumprob), deriv (first derivative of cumprob), and deriv2 (second derivative of cumprob)
deviance	-2 log likelihoods (counting penalty components) When an offset variable is present, three deviances are computed: for intercept(s) only, for intercepts+offset, and for intercepts+offset+predictors. When there is no offset variable, the vector contains deviances for the intercept(s)-only model and the model with intercept(s) and predictors.
non.slopes	number of intercepts in model
interceptRef	the index of the middle (median) intercept used in computing the linear predictor and var
penalty	see lrm
penalty.matrix	the penalty matrix actually used in the estimation
info.matrix	a sparse matrix representation of type matrix.csr from the SparseM package. This allows the full information matrix with all intercepts to be stored efficiently, and matrix operations using the Cholesky decomposition to be fast. link{vcov.orm} uses this information to compute the covariance matrix for intercepts other than the middle one.

Note

When creating ordinal levels for non-integer numeric Y, this function does not use the unique function because it was found to be non-reproducible across hardware platforms. Instead, intercepts are mapped to integers obtained from rounding original Y levels after multiplying by 1e7 (by default). When using the ExProb or Survival functions to compute exceedance probabilities or survival curves, when the user wishes to obtain probabilities at select non-integer y values, the only way to guarantee the correct mapping to intercepts is to request estimates at the "official" levels of Y stored in the yunique object found in the fit object. Users may opt instead to replace original data with e.g. 1e-7 * round(Y * 1e7) before model fitting.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com
For the Quantile function:
Qi Liu and Shengxin Tu
Department of Biostatistics, Vanderbilt University

References

Sall J: A monotone regression smoother based on ordinal cumulative logistic regression, 1991.

Le Cessie S, Van Houwelingen JC: Ridge estimators in logistic regression. Applied Statistics 41:191–201, 1992.

Verweij PJM, Van Houwelingen JC: Penalized likelihood in Cox regression. Stat in Med 13:2427–2436, 1994.

Gray RJ: Flexible methods for analyzing survival data using splines, with applications to breast cancer prognosis. JASA 87:942–951, 1992.

Shao J: Linear model selection by cross-validation. JASA 88:486–494, 1993.

Verweij PJM, Van Houwelingen JC: Crossvalidation in survival analysis. Stat in Med 12:2305–2314, 1993.

Harrell FE: Model uncertainty, penalization, and parsimony. Available from https://hbiostat.org/talks/iscb98.pdf.

See Also

orm.fit, predict.orm, solve, ordESS, rms.trans, rms, polr, latex.orm, vcov.orm, num.intercepts, residuals.orm, na.delete, na.detail.response, pentrace, rmsMisc, vif, predab.resample, validate.orm, calibrate, Mean.orm, gIndex, prModFit

Examples

```
require(ggplot2)
set.seed(1)
n <- 100
y <- round(runif(n), 2)
x1 <- sample(c(-1,0,1), n, TRUE)
x2 <- sample(c(-1,0,1), n, TRUE)
f <- lrm(y ~ x1 + x2, eps=1e-5)
g <- orm(y ~ x1 + x2, eps=1e-5)
max(abs(coef(g) - coef(f)))
w <- vcov(g, intercepts='all') / vcov(f) - 1
max(abs(w))

set.seed(1)
n <- 300
x1 <- c(rep(0,150), rep(1,150))
y <- rnorm(n) + 3*x1
g <- orm(y ~ x1)
g
k <- coef(g)
i <- num.intercepts(g)
h <- orm(y ~ x1, family='probit')
ll <- orm(y ~ x1, family='loglog')
cll <- orm(y ~ x1, family='cloglog')
cau <- orm(y ~ x1, family='cauchit')
x <- 1:i
z <- list(logistic=list(x=x, y=coef(g)[1:i]),
          probit  =list(x=x, y=coef(h)[1:i]),
          loglog  =list(x=x, y=coef(ll)[1:i]),
          cloglog =list(x=x, y=coef(cll)[1:i]))
labcurve(z, pl=TRUE, col=1:4, ylab='Intercept')

tapply(y, x1, mean)
m <- Mean(g)
m(w <- k[1] + k['x1']*c(0,1))
mh <- Mean(h)
```

```
wh <- coef(h)[1] + coef(h)['x1']*c(0,1)
mh(wh)

qu <- Quantile(g)
# Compare model estimated and empirical quantiles
cq <- function(y) {
   cat(qu(.1, w), tapply(y, x1, quantile, probs=.1), '\n')
   cat(qu(.5, w), tapply(y, x1, quantile, probs=.5), '\n')
   cat(qu(.9, w), tapply(y, x1, quantile, probs=.9), '\n')
   }
cq(y)

# Try on log-normal model
g <- orm(exp(y) ~ x1)
g
k <- coef(g)
plot(k[1:i])
m <- Mean(g)
m(w <- k[1] + k['x1']*c(0,1))
tapply(exp(y), x1, mean)

qu <- Quantile(g)
cq(exp(y))

# Compare predicted mean with ols for a continuous x
set.seed(3)
n <- 200
x1 <- rnorm(n)
y <- x1 + rnorm(n)
dd <- datadist(x1); options(datadist='dd')
f <- ols(y ~ x1)
g <- orm(y ~ x1, family='probit')
h <- orm(y ~ x1, family='logistic')
w <- orm(y ~ x1, family='cloglog')
mg <- Mean(g); mh <- Mean(h); mw <- Mean(w)
r <- rbind(ols      = Predict(f, conf.int=FALSE),
           probit   = Predict(g, conf.int=FALSE, fun=mg),
           logistic = Predict(h, conf.int=FALSE, fun=mh),
           cloglog  = Predict(w, conf.int=FALSE, fun=mw))
plot(r, groups='.set.')

# Compare predicted 0.8 quantile with quantile regression
qu <- Quantile(g)
qu80 <- function(lp) qu(.8, lp)
f <- Rq(y ~ x1, tau=.8)
r <- rbind(probit   = Predict(g, conf.int=FALSE, fun=qu80),
           quantreg = Predict(f, conf.int=FALSE))
plot(r, groups='.set.')

# Verify transformation invariance of ordinal regression
ga <- orm(exp(y) ~ x1, family='probit')
qua <- Quantile(ga)
qua80 <- function(lp) log(qua(.8, lp))
```

```
r <- rbind(logprobit = Predict(ga, conf.int=FALSE, fun=qua80),
           probit    = Predict(g,  conf.int=FALSE, fun=qu80))
plot(r, groups='.set.')

# Try the same with quantile regression.  Need to transform x1
fa <- Rq(exp(y) ~ rcs(x1,5), tau=.8)
r <- rbind(qr    = Predict(f, conf.int=FALSE),
           logqr = Predict(fa, conf.int=FALSE, fun=log))
plot(r, groups='.set.')

# Make a plot of Pr(Y >= y) vs. a continuous covariate for 3 levels
# of y and also against a binary covariate
set.seed(1)
n <- 1000
age <- rnorm(n, 50, 15)
sex <- sample(c('m', 'f'), 1000, TRUE)
Y   <- runif(n)
dd  <- datadist(age, sex); options(datadist='dd')
f <- orm(Y ~ age + sex)
# Use ExProb function to derive an R function to compute
# P(Y >= y | X)
ex  <- ExProb(f)
ex1 <- function(x) ex(x, y=0.25)
ex2 <- function(x) ex(x, y=0.5)
ex3 <- function(x) ex(x, y=0.75)
p1  <- Predict(f, age, sex, fun=ex1)
p2  <- Predict(f, age, sex, fun=ex2)
p3  <- Predict(f, age, sex, fun=ex3)
p   <- rbind('P(Y >= 0.25)' = p1,
             'P(Y >= 0.5)'  = p2,
             'P(Y >= 0.75)' = p3)
ggplot(p)

# Make plot with two curves (by sex) with y on the x-axis, and
# estimated P(Y >= y | sex, age=median) on the y-axis
ys <- seq(min(Y), max(Y), length=100)
g <- function(sx) as.vector(ex(y=ys, Predict(f, sex=sx)$yhat)$prob)

d  <- rbind(data.frame(sex='m', y=ys, p=g('m')),
            data.frame(sex='f', y=ys, p=g('f')))
ggplot(d, aes(x=y, y=p, color=sex)) + geom_line() +
  ylab(expression(P(Y >= y))) +
  guides(color=guide_legend(title='Sex')) +
  theme(legend.position='bottom')

options(datadist=NULL)
## Not run:
## Simulate power and type I error for orm logistic and probit regression
## for likelihood ratio, Wald, and score chi-square tests, and compare
## with t-test
require(rms)
set.seed(5)
nsim <- 2000
```

```
r <- NULL
for(beta in c(0, .4)) {
  for(n in c(10, 50, 300)) {
    cat('beta=', beta, '  n=', n, '\n\n')
    plogistic <- pprobit <- plogistics <- pprobits <- plogisticw <-
      pprobitw <- ptt <- numeric(nsim)
    x <- c(rep(0, n/2), rep(1, n/2))
    pb <- setPb(nsim, every=25, label=paste('beta=', beta, '  n=', n))
    for(j in 1:nsim) {
      pb(j)
      y <- beta*x + rnorm(n)
      tt <- t.test(y ~ x)
      ptt[j] <- tt$p.value
      f <- orm(y ~ x)
      plogistic[j]  <- f$stats['P']
      plogistics[j] <- f$stats['Score P']
      plogisticw[j] <- 1 - pchisq(coef(f)['x']^2 / vcov(f)[2,2], 1)
      f <- orm(y ~ x, family='probit')
      pprobit[j]  <- f$stats['P']
      pprobits[j] <- f$stats['Score P']
      pprobitw[j] <- 1 - pchisq(coef(f)['x']^2 / vcov(f)[2,2], 1)
    }
    if(beta == 0) plot(ecdf(plogistic))
    r <- rbind(r, data.frame(beta         = beta, n=n,
                             ttest        = mean(ptt        < 0.05),
                             logisticlr   = mean(plogistic  < 0.05),
                             logisticscore= mean(plogistics < 0.05),
                             logisticwald = mean(plogisticw < 0.05),
                             probit       = mean(pprobit    < 0.05),
                             probitscore  = mean(pprobits   < 0.05),
                             probitwald   = mean(pprobitw   < 0.05)))
  }
}
print(r)
#  beta   n  ttest logisticlr logisticscore logisticwald probit probitscore probitwald
#1  0.0  10 0.0435     0.1060        0.0655        0.043 0.0920      0.0920     0.0820
#2  0.0  50 0.0515     0.0635        0.0615        0.060 0.0620      0.0620     0.0620
#3  0.0 300 0.0595     0.0595        0.0590        0.059 0.0605      0.0605     0.0605
#4  0.4  10 0.0755     0.1595        0.1070        0.074 0.1430      0.1430     0.1285
#5  0.4  50 0.2950     0.2960        0.2935        0.288 0.3120      0.3120     0.3120
#6  0.4 300 0.9240     0.9215        0.9205        0.920 0.9230      0.9230     0.9230

## End(Not run)
```

`orm.fit` *Ordinal Regression Model Fitter*

Description

Fits ordinal cumulative probability models for continuous or ordinal response variables, efficiently allowing for a large number of intercepts by capitalizing on the information matrix being sparse.

Five different distribution functions are implemented, with the default being the logistic (yielding the proportional odds model). Penalized estimation and weights are also implemented, as in '[lrm.fit()]'. The optimization method is Newton-Raphson with step-halving, or the Levenberg-Marquart method. The latter has been shown to converge better when there are large offsets. Execution time is is fast even for hundreds of thousands of intercepts. The limiting factor is the number of intercepts times the number of columns of `x`.

Usage

```
orm.fit(x=NULL, y, family=c("logistic","probit","loglog","cloglog","cauchit"),
        offset, initial, opt_method=c('NR', 'LM'),
        maxit=30L, eps=5e-4, gradtol=0.001, abstol=1e10,
        minstepsize=0.01, tol=.Machine$double.eps, trace=FALSE,
     penalty.matrix=NULL, weights=NULL, normwt=FALSE, scale=FALSE, mscore=FALSE,
        inclpen=TRUE, y.precision = 7, compstats=TRUE, onlydata=FALSE, ...)
```

Arguments

`x`	design matrix with no column for an intercept
`y`	response vector, numeric, factor, or character. The ordering of levels is assumed from `factor(y)`.
`family`	a character value specifying the distribution family, corresponding to logistic (the default), Gaussian, Cauchy, Gumbel maximum ($exp(-exp(-x))$; extreme value type I), and Gumbel minimum ($1 - exp(-exp(x))$) distributions. These are the cumulative distribution functions assumed for $Prob[Y \geq y \mid X]$. The `family` argument can be an unquoted or a quoted string, e.g. `family=loglog` or `family="loglog"`. To use a built-in family, the string must be one of the following corresponding to the previous list: `logistic`, `probit`, `loglog`,`cloglog`, `cauchit`.
`offset`	optional numeric vector containing an offset on the logit scale
`initial`	vector of initial parameter estimates, beginning with the intercepts. If `initial` is not specified, the function computes the overall score χ^2 test for the global null hypothesis of no regression. `initial` is padded to the right with zeros for the regression coefficients, if needed. When censoring is present, `initial` can also be a list with elements `time` and `surv` from the `npsurv` attribute of the `y` element of a previous fit. This is useful when bootstrapping, for example.
`opt_method`	set to `"LM"` to use Levenberg-Marquardt instead of the default Newton-Raphson
`maxit`	maximum no. iterations (default=`30`).
`eps`	difference in $-2log$ likelihood for declaring convergence. Default is `.0005`. This handles the case where the initial estimates are MLEs, to prevent endless step-halving.
`gradtol`	maximum absolute gradient before convergence can be declared. `gradtol` is automatically scaled by n / 1000 since the gradient is proportional to the sample size.
`abstol`	maximum absolute change in parameter estimates from one iteration to the next before convergence can be declared; by default has no effect

minstepsize	used to specify when to abandon step-halving
tol	Singularity criterion. Default is typically 2e-16
trace	set to TRUE to print -2 log likelihood, step-halving fraction, change in -2 log likelihood, maximum absolute value of first derivative, and max absolute change in parameter estimates at each iteration.
penalty.matrix	a self-contained ready-to-use penalty matrix - seelrm
weights	a vector (same length as y) of possibly fractional case weights
normwt	set to TRUE to scale weights so they sum to n, the length of y; useful for sample surveys as opposed to the default of frequency weighting
mscore	set to TRUE to compute the sparse score matrix and store its elements as a list mscore
scale	set to TRUE to subtract column means and divide by column standard deviations of x before fitting, and to back-solve for the un-normalized covariance matrix and regression coefficients. This can sometimes make the model converge for very large sample sizes where for example spline or polynomial component variables create scaling problems leading to loss of precision when accumulating sums of squares and crossproducts.
inclpen	set to FALSE to not include the penalty matrix in the Hessian when the Hessian is being computed on transformed x, vs. adding the penalty after back-transforming. This should not matter.
y.precision	When 'y' is numeric, values may need to be rounded to avoid unpredictable behavior with unique() with floating-point numbers. Default is to 7 decimal places.
compstats	set to FALSE to prevent the calculation of the vector of model statistics
onlydata	set to TRUE to return the data used in model fitting as a list, without fitting the model
...	ignored

Value

a list with the following components, not counting all the components produced by 'orm.fit':

call	calling expression
freq	table of frequencies for y in order of increasing y
yunique	vector of sorted unique values of y
stats	vector with the following elements: number of observations used in the fit, number of unique y values, median y from among the observations used in the fit, maximum absolute value of first derivative of log likelihood, model likelihood ratio chi-square, d.f., P-value, score chi-square and its P-value, Spearman's ρ rank correlation between linear predictor and y (if there is no censoring), Somers' Dxy rank correlation (if there is no censoring or only right censoring),) the Nagelkerke R^2 index, other R^2 measures, the g-index, gr (the g-index on the ratio scale), and pdm (the mean absolute difference between 0.5 and the estimated probability that $y \geq$ the marginal median). When penalty.matrix is present, the χ^2, d.f., and P-value are not corrected for the effective d.f.

`fail` set to `TRUE` if convergence failed (and `maxit>1`)

`coefficients` estimated parameters

`family`, `famfunctions`
see `orm`

`deviance` -2 log likelihoods. When an offset variable is present, three deviances are computed: for intercept(s) only, for intercepts+offset, and for intercepts+offset+predictors. When there is no offset variable, the vector contains deviances for the intercept(s)-only model and the model with intercept(s) and predictors.

`lpe` vector of per-observation likelihood probability elements. An observation's contribution to the log likelihood is the log of `lpe`.

`non.slopes` number of intercepts in model

`interceptRef` the index of the middle (median) intercept used in computing the linear predictor and `var`

`linear.predictors`
the linear predictor using the first intercept

`penalty.matrix` see above

`info.matrix` see `orm`

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

`orm`, `lrm`, `glm`, `gIndex`, `solve`, `recode2integer`

Examples

```
#Fit an additive logistic model containing numeric predictors age,
#blood.pressure, and sex, assumed to be already properly coded and
#transformed
#
# fit <- orm.fit(cbind(age,blood.pressure,sex), death)
```

`pentrace` *Trace AIC and BIC vs. Penalty*

Description

For an ordinary unpenalized fit from `lrm`, `orm`, or `ols` and for a vector or list of penalties, fits a series of logistic or linear models using penalized maximum likelihood estimation, and saves the effective degrees of freedom, Akaike Information Criterion (AIC), Schwarz Bayesian Information Criterion (BIC), and Hurvich and Tsai's corrected AIC (AIC_c). Optionally `pentrace` can use the `nlminb` function to solve for the optimum penalty factor or combination of factors penalizing different kinds of terms in the model. The `effective.df` function prints the original and effective degrees of freedom for a penalized fit or for an unpenalized fit and the best penalization determined from a previous invocation of `pentrace` if `method="grid"` (the default). The effective d.f. is computed separately for each class of terms in the model (e.g., interaction, nonlinear). A `plot` method exists to plot the results, and a `print` method exists to print the most pertinent components. Both AIC and BIC may be plotted if there is only one penalty factor type specified in `penalty`. Otherwise, the first two types of penalty factors are plotted, showing only the AIC.

Usage

```
pentrace(fit, penalty, penalty.matrix,
         method=c('grid','optimize'),
         which=c('aic.c','aic','bic'), target.df=NULL,
         fitter, pr=FALSE, tol=.Machine$double.eps,
         keep.coef=FALSE, complex.more=TRUE, verbose=FALSE, maxit=20,
         subset, noaddzero=FALSE, ...)

effective.df(fit, object)

## S3 method for class 'pentrace'
print(x, ...)

## S3 method for class 'pentrace'
plot(x, method=c('points','image'),
     which=c('effective.df','aic','aic.c','bic'), pch=2, add=FALSE,
     ylim, ...)
```

Arguments

`fit` a result from `lrm`, `orm`, or `ols` with `x=TRUE, y=TRUE` and without using `penalty` or `penalty.matrix` (or optionally using penalization in the case of `effective.df`)

`penalty` can be a vector or a list. If it is a vector, all types of terms in the model will be penalized by the same amount, specified by elements in `penalty`, with a penalty of zero automatically added. `penalty` can also be a list in the format documented in the `lrm` function, except that elements of the list can be vectors. The `expand.grid` function is invoked by `pentrace` to generate all possible combinations of penalties. For example, specifying `penalty=list(simple=1:2, nonlinear=1:3)` will generate 6 combinations to try, so that the analyst can attempt to determine whether penalizing more complex terms in the model more than the linear or categorical variable terms will be beneficial. If `complex.more=TRUE`, it is assumed that the variables given in `penalty` are listed in order from less

complex to more complex. With `method="optimize"` `penalty` specifies an initial guess for the penalty or penalties. If all term types are to be equally penalized, `penalty` should be a single number, otherwise it should be a list containing single numbers as elements, e.g., `penalty=list(simple=1, nonlinear=2)`. Experience has shown that the optimization algorithm is more likely to find a reasonable solution when the starting value specified in `penalty` is too large rather than too small.

`object` an object returned by `pentrace`. For `effective.df`, `object` can be omitted if the `fit` was penalized.

`penalty.matrix` see `lrm`

`method` The default is `method="grid"` to print various indexes for all combinations of penalty parameters given by the user. Specify `method="optimize"` to have `pentrace` use `nlminb` to solve for the combination of penalty parameters that gives the maximum value of the objective named in `which`, or, if `target.df` is given, to find the combination that yields `target.df` effective total degrees of freedom for the model. When `target.df` is specified, `method` is set to `"optimize"` automatically. For `plot.pentrace` this parameter applies only if more than one penalty term-type was used. The default is to use open triangles whose sizes are proportional to the ranks of the AICs, plotting the first two penalty factors respectively on the x and y axes. Use `method="image"` to plot an image plot.

`which` the objective to maximize for either `method`. Default is `"aic.c"` (corrected AIC). For `plot.pentrace`, `which` is a vector of names of criteria to show; default is to plot all 4 types, with effective d.f. in its own separate plot

`target.df` applies only to `method="optimize"`. See `method`. `target.df` makes sense mainly when a single type of penalty factor is specified.

`fitter` a fitting function. Default is `lrm.fit` (`lm.pfit` is always used for `ols`).

`pr` set to `TRUE` to print intermediate results

`tol` tolerance for declaring a matrix singular (see `lrm.fit, solvet`)

`keep.coef` set to `TRUE` to store matrix of regression coefficients for all the fits (corresponding to increasing values of `penalty`) in object `Coefficients` in the returned list. Rows correspond to penalties, columns to regression parameters.

`complex.more` By default if `penalty` is a list, combinations of penalties for which complex terms are penalized less than less complex terms will be dropped after `expand.grid` is invoked. Set `complex.more=FALSE` to allow more complex terms to be penalized less. Currently this option is ignored for `method="optimize"`.

`verbose` set to `TRUE` to print number of intercepts and sum of effective degrees of freedom

`maxit` maximum number of iterations to allow in a model fit (default=12). This is passed to the appropriate fitter function with the correct argument name. Increase `maxit` if you had to when fitting the original unpenalized model.

`subset` a logical or integer vector specifying rows of the design and response matrices to subset in fitting models. This is most useful for bootstrapping `pentrace` to see if the best penalty can be estimated with little error so that variation due to selecting the optimal penalty can be safely ignored when bootstrapping standard errors of regression coefficients and measures of predictive accuracy. See an example below.

noaddzero	set to TRUE to not add an unpenalized model to the list of models to fit
x	a result from pentrace
pch	used for method="points"
add	set to TRUE to add to an existing plot. In that case, the effective d.f. plot is not re-drawn, but the AIC/BIC plot is added to.
ylim	2-vector of y-axis limits for plots other than effective d.f.
...	other arguments passed to plot, lines, or image, or to the fitter

Value

a list of class "pentrace" with elements penalty, df, objective, fit, var.adj, diag, results.all, and optionally Coefficients. The first 6 elements correspond to the fit that had the best objective as named in the which argument, from the sequence of fits tried. Here fit is the fit object from fitter which was a penalized fit, diag is the diagonal of the matrix used to compute the effective d.f., and var.adj is Gray (1992) Equation 2.9, which is an improved covariance matrix for the penalized beta. results.all is a data frame whose first few variables are the components of penalty and whose other columns are df, aic, bic, aic.c. results.all thus contains a summary of results for all fits attempted. When method="optimize", only two components are returned: penalty and objective, and the object does not have a class.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

References

Gray RJ: Flexible methods for analyzing survival data using splines, with applications to breast cancer prognosis. JASA 87:942–951, 1992.

Hurvich CM, Tsai, CL: Regression and time series model selection in small samples. Biometrika 76:297–307, 1989.

See Also

lrm, orm, ols, solvet, rmsMisc, image

Examples

```
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
```

```
  (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male'))
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)


f <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)),
         x=TRUE, y=TRUE)
p <- pentrace(f, seq(.2,1,by=.05))
plot(p)
p$diag      # may learn something about fractional effective d.f.
            # for each original parameter
pentrace(f, list(simple=c(0,.2,.4), nonlinear=c(0,.2,.4,.8,1)))


# Bootstrap pentrace 5 times, making a plot of corrected AIC plot with 5 reps
n <- nrow(f$x)
plot(pentrace(f, seq(.2,1,by=.05)), which='aic.c',
     col=1, ylim=c(30,120)) #original in black
for(j in 1:5)
  plot(pentrace(f, seq(.2,1,by=.05), subset=sample(n,n,TRUE)),
       which='aic.c', col=j+1, add=TRUE)


# Find penalty giving optimum corrected AIC.  Initial guess is 1.0
# Not implemented yet
# pentrace(f, 1, method='optimize')


# Find penalty reducing total regression d.f. effectively to 5
# pentrace(f, 1, target.df=5)


# Re-fit with penalty giving best aic.c without differential penalization
f <- update(f, penalty=p$penalty)
effective.df(f)
```

`plot.contrast.rms` *plot.contrast.rms*

Description

Plot Bayesian Contrast Posterior Densities

Usage

```
## S3 method for class 'contrast.rms'
plot(
  x,
  bivar = FALSE,
  bivarmethod = c("ellipse", "kernel"),
```

```
  prob = 0.95,
  which = c("both", "diff", "ind"),
  nrow = NULL,
  ncol = NULL,
  ...
)
```

Arguments

x	the result of `contrast.rms`
bivar	set to `TRUE` to plot 2-d posterior density contour
bivarmethod	see `rmsb::pdensityContour()`
prob	posterior coverage probability for HPD interval or 2-d contour
which	applies when plotting the result of `contrast(..., fun=)`, defaulting to showing the posterior density of both estimates plus their difference. Set to `"ind"` to only show the two individual densities or `"diff"` to only show the posterior density for the differences.
nrow	for `ggplot2::facet_wrap()`
ncol	likewise
...	unused

Details

If there are exactly two contrasts and `bivar=TRUE` plots an elliptical or kernal (based on `bivarmethod` posterior density contour with probability `prob`). Otherwise plots a series of posterior densities of contrasts along with HPD intervals, posterior means, and medians. When the result being plotted comes from `contrast` with `fun=` specified, both the two individual estimates and their difference are plotted.

Value

`ggplot2` object

Author(s)

Frank Harrell

`plot.Predict` *Plot Effects of Variables Estimated by a Regression Model Fit*

Description

Uses `lattice` graphics to plot the effect of one or two predictors on the linear predictor or X beta scale, or on some transformation of that scale. The first argument specifies the result of the `Predict` function. The predictor is always plotted in its original coding. `plot.Predict` uses the `xYplot` function unless `formula` is omitted and the x-axis variable is a factor, in which case it reverses the x- and y-axes and uses the `Dotplot` function.

If `data` is given, a rug plot is drawn showing the location/density of data values for the x-axis variable. If there is a `groups` (superposition) variable that generated separate curves, the data density specific to each class of points is shown. This assumes that the second variable was a factor variable. The rug plots are drawn by `scat1d`. When the same predictor is used on all x-axes, and multiple panels are drawn, you can use `subdata` to specify an expression to subset according to other criteria in addition.

To plot effects instead of estimates (e.g., treatment differences as a function of interacting factors) see `contrast.rms` and `summary.rms`.

`pantext` creates a `lattice` panel function for including text such as that produced by `print.anova.rms` inside a panel or in a base graphic.

Usage

```
## S3 method for class 'Predict'
plot(x, formula, groups=NULL,
     cond=NULL, varypred=FALSE, subset,
     xlim, ylim, xlab, ylab,
     data=NULL, subdata, anova=NULL, pval=FALSE, cex.anova=.85,
     col.fill=gray(seq(.825, .55, length=5)),
     adj.subtitle, cex.adj, cex.axis, perim=NULL, digits=4, nlevels=3,
     nlines=FALSE, addpanel, scat1d.opts=list(frac=0.025, lwd=0.3),
     type=NULL, yscale=NULL, scaletrans=function(z) z, ...)

pantext(object, x, y, cex=.5, adj=0, fontfamily="Courier", lattice=TRUE)
```

Arguments

`x`	a data frame created by `Predict`, or for `pantext` the x-coordinate for text
`formula`	the right hand side of a `lattice` formula reference variables in data frame `x`. You may not specify `formula` if you varied multiple predictors separately when calling `Predict`. Otherwise, when `formula` is not given, `plot.Predict` constructs one from information in `x`.
`groups`	an optional name of one of the variables in `x` that is to be used as a grouping (superpositioning) variable. Note that `groups` does not contain the groups data as is customary in `lattice`; it is only a single character string specifying the name of the grouping variable.
`cond`	when plotting effects of different predictors, `cond` is a character string that specifies a single variable name in `x` that can be used to form panels. Only applies if using `rbind` to combine several `Predict` results.

varypred — set to TRUE if x is the result of passing multiple Predict results, that represent different predictors, to rbind.Predict. This will cause the .set. variable created by rbind to be copied to the .predictor. variable.

subset — a subsetting expression for restricting the rows of x that are used in plotting. For example, predictions may have been requested for males and females but one wants to plot only females.

xlim — This parameter is seldom used, as limits are usually controlled with Predict. One reason to use xlim is to plot a factor variable on the x-axis that was created with the cut2 function with the levels.mean option, with val.lev=TRUE specified to plot.Predict. In this case you may want the axis to have the range of the original variable values given to cut2 rather than the range of the means within quantile groups.

ylim — Range for plotting on response variable axis. Computed by default.

xlab — Label for x-axis. Default is one given to asis, rcs, etc., which may have been the "label" attribute of the variable.

ylab — Label for y-axis. If fun is not given, default is "log Odds" for lrm, "log Relative Hazard" for cph, name of the response variable for ols, TRUE or log(TRUE) for psm, or "X * Beta" otherwise.

data — a data frame containing the original raw data on which the regression model were based, or at least containing the x-axis and grouping variable. If data is present and contains the needed variables, the original data are added to the graph in the form of a rug plot using scat1d.

subdata — if data is specified, an expression to be evaluated in the data environment that evaluates to a logical vector specifying which observations in data to keep. This will be intersected with the criterion for the groups variable. Example: if conditioning on two paneling variables using |a*b you can specify subdata=b==levels(b)[which.packet()
where the 2 comes from the fact that b was listed second after the vertical bar (this assumes b is a factor in data. Another example: subdata=sex==c('male','female')[current.

anova — an object returned by anova.rms. If anova is specified, the overall test of association for predictor plotted is added as text to each panel, located at the spot at which the panel is most empty unless there is significant empty space at the top or bottom of the panel; these areas are given preference.

pval — specify pval=TRUE for anova to include not only the test statistic but also the P-value

cex.anova — character size for the test statistic printed on the panel

col.fill — a vector of colors used to fill confidence bands for successive superposed groups. Default is inceasingly dark gray scale.

adj.subtitle — Set to FALSE to suppress subtitling the graph with the list of settings of non-graphed adjustment values.

cex.adj — cex parameter for size of adjustment settings in subtitles. Default is 0.75 times par("cex").

cex.axis — cex parameter for x-axis tick labels

perim — perim specifies a function having two arguments. The first is the vector of values of the first variable that is about to be plotted on the x-axis. The second

argument is the single value of the variable representing different curves, for the current curve being plotted. The function's returned value must be a logical vector whose length is the same as that of the first argument, with values TRUE if the corresponding point should be plotted for the current curve, FALSE otherwise. See one of the latter examples. If a predictor is not specified to plot, NULL is passed as the second argument to perim, although it makes little sense to use perim when the same perim is used for multiple predictors.

digits — Controls how numeric variables used for panel labels are formatted. The default is 4 significant digits.

nlevels — when groups and formula are not specified, if any panel variable has nlevels or fewer values, that variable is converted to a groups (superpositioning) variable. Set nlevels=0 to prevent this behavior. For other situations, a numeric x-axis variable with nlevels or fewer unique values will cause a dot plot to be drawn instead of an x-y plot.

nlines — If formula is given, you can set nlines to TRUE to convert the x-axis variable to a factor and then to an integer. Points are plotted at integer values on the x-axis but labeled with category levels. Points are connected by lines.

addpanel — an additional panel function to call along with panel functions used for xYplot and Dotplot displays

scat1d.opts — a list containing named elements that specifies parameters to scat1d when data is given. The col parameter is usually derived from other plotting information and not specified by the user.

type — a value ("l","p","b") to override default choices related to showing or connecting points. Especially useful for discrete x coordinate variables.

yscale — a lattice scale list for the y-axis to be added to what is automatically generated for the x-axis. Example: yscale=list(at=c(.005,.01,.05),labels=format(c(.005,.01,.05)
See xyplot

scaletrans — a function that operates on the scale object created by plot.Predict to produce a modified scale object that is passed to the lattice graphics function. This is useful for adding other scales options or for changing the x-axis limits for one predictor.

... — extra arguments to pass to xYplot or Dotplot. Some useful ones are label.curves and abline. Set label.curves to FALSE to suppress labeling of separate curves. Default is TRUE, which causes labcurve to be invoked to place labels at positions where the curves are most separated, labeling each curve with the full curve label. Set label.curves to a list to specify options to labcurve, e.g., label.curves= list(method="arrow",cex=.8). These option names may be abbreviated in the usual way arguments are abbreviated. Use for example label.curves=list(keys=letters[1:5]) to draw single lower case letters on 5 curves where they are most separated, and automatically position a legend in the most empty part of the plot. The col, lty, and lwd parameters are passed automatically to labcurve although they may be overridden here. It is also useful to use ...to pass lattice graphics parameters, e.g. par.settings=list(axis.text=list(cex=1.2), par.ylab.text=list(col='blue',cex=.9),par

object — an object having a print method

y	y-coordinate for placing text in a lattice panel or on a base graphics plot
cex	character expansion size for pantext
adj	text justification. Default is left justified.
fontfamily	font family for pantext. Default is "Courier" which will line up columns of a table.
lattice	set to FALSE to use text instead of ltext in the function generated by pantext, to use base graphics

Details

When a groups (superpositioning) variable was used, you can issue the command Key(...) after printing the result of plot.Predict, to draw a key for the groups.

Value

a lattice object ready to print for rendering.

Note

If plotting the effects of all predictors you can reorder the panels using for example p <- Predict(fit); p$.predictor. <-factor(p$.predictor., v) where v is a vector of predictor names specified in the desired order.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Fox J, Hong J (2009): Effect displays in R for multinomial and proportional-odds logit models: Extensions to the effects package. J Stat Software 32 No. 1.

See Also

Predict, ggplot.Predict, link{plotp.Predict}, rbind.Predict, datadist, predictrms, anova.rms, contrast.rms, summary.rms, rms, rmsMisc, labcurve, scat1d, xYplot, Overview

Examples

```
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
label(age)            <- 'Age'      # label is in Hmisc
label(cholesterol)    <- 'Total Cholesterol'
```

```
label(blood.pressure) <- 'Systolic Blood Pressure'
label(sex)            <- 'Sex'
units(cholesterol)    <- 'mg/dl'   # uses units.default in Hmisc
units(blood.pressure) <- 'mmHg'

# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
  (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male'))
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)

ddist <- datadist(age, blood.pressure, cholesterol, sex)
options(datadist='ddist')

fit <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)),
               x=TRUE, y=TRUE)
an <- anova(fit)
# Plot effects of all 4 predictors with test statistics from anova, and P
plot(Predict(fit), anova=an, pval=TRUE)
plot(Predict(fit), data=llist(blood.pressure,age))
                         # rug plot for two of the predictors

p <- Predict(fit, name=c('age','cholesterol'))   # Make 2 plots
plot(p)

p <- Predict(fit, age=seq(20,80,length=100), sex, conf.int=FALSE)
                         # Plot relationship between age and log
                         # odds, separate curve for each sex,
plot(p, subset=sex=='female' | age > 30)
# No confidence interval, suppress estimates for males <= 30

p <- Predict(fit, age, sex)
plot(p, label.curves=FALSE, data=llist(age,sex))
                         # use label.curves=list(keys=c('a','b'))'
                         # to use 1-letter abbreviations
                         # data= allows rug plots (1-dimensional scatterplots)
                         # on each sex's curve, with sex-
                         # specific density of age
                         # If data were in data frame could have used that
p <- Predict(fit, age=seq(20,80,length=100), sex='male', fun=plogis)
                         # works if datadist not used
plot(p, ylab=expression(hat(P)))
                         # plot predicted probability in place of log odds

per <- function(x, y) x >= 30
plot(p, perim=per)       # suppress output for age < 30 but leave scale alone

# Take charge of the plot setup by specifying a lattice formula
p <- Predict(fit, age, blood.pressure=c(120,140,160),
             cholesterol=c(180,200,215), sex)
plot(p, ~ age | blood.pressure*cholesterol, subset=sex=='male')
# plot(p, ~ age | cholesterol*blood.pressure, subset=sex=='female')
# plot(p, ~ blood.pressure|cholesterol*round(age,-1), subset=sex=='male')
```

```
plot(p)

# Plot the age effect as an odds ratio
# comparing the age shown on the x-axis to age=30 years

ddist$limits$age[2] <- 30    # make 30 the reference value for age
# Could also do: ddist$limits["Adjust to","age"] <- 30
fit <- update(fit)   # make new reference value take effect
p <- Predict(fit, age, ref.zero=TRUE, fun=exp)
plot(p, ylab='Age=x:Age=30 Odds Ratio',
     abline=list(list(h=1, lty=2, col=2), list(v=30, lty=2, col=2)))

# Compute predictions for three predictors, with superpositioning or
# conditioning on sex, combined into one graph

p1 <- Predict(fit, age, sex)
p2 <- Predict(fit, cholesterol, sex)
p3 <- Predict(fit, blood.pressure, sex)
p <- rbind(age=p1, cholesterol=p2, blood.pressure=p3)
plot(p, groups='sex', varypred=TRUE, adj.subtitle=FALSE)
plot(p, cond='sex', varypred=TRUE, adj.subtitle=FALSE)

## Not run:
# For males at the median blood pressure and cholesterol, plot 3 types
# of confidence intervals for the probability on one plot, for varying age
ages <- seq(20, 80, length=100)
p1 <- Predict(fit, age=ages, sex='male', fun=plogis)  # standard pointwise
p2 <- Predict(fit, age=ages, sex='male', fun=plogis,
              conf.type='simultaneous')               # simultaneous
p3 <- Predict(fit, age=c(60,65,70), sex='male', fun=plogis,
              conf.type='simultaneous')               # simultaneous 3 pts
# The previous only adjusts for a multiplicity of 3 points instead of 100
f <- update(fit, x=TRUE, y=TRUE)
g <- bootcov(f, B=500, coef.reps=TRUE)
p4 <- Predict(g, age=ages, sex='male', fun=plogis)    # bootstrap percentile
p <- rbind(Pointwise=p1, 'Simultaneous 100 ages'=p2,
           'Simultaneous     3 ages'=p3, 'Bootstrap nonparametric'=p4)
xYplot(Cbind(yhat, lower, upper) ~ age, groups=.set.,
       data=p, type='l', method='bands', label.curve=list(keys='lines'))

## End(Not run)

# Plots for a parametric survival model
require(survival)
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('Male','Female'), n,
              rep=TRUE, prob=c(.6, .4)))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
t <- -log(runif(n))/h
```

```
label(t) <- 'Follow-up Time'
e <- ifelse(t<=cens,1,0)
t <- pmin(t, cens)
units(t) <- "Year"
ddist <- datadist(age, sex)
Srv <- Surv(t,e)


# Fit log-normal survival model and plot median survival time vs. age
f <- psm(Srv ~ rcs(age), dist='lognormal')
med <- Quantile(f)       # Creates function to compute quantiles
                         # (median by default)
p <- Predict(f, age, fun=function(x) med(lp=x))
plot(p, ylab="Median Survival Time")
# Note: confidence intervals from this method are approximate since
# they don't take into account estimation of scale parameter


# Fit an ols model to log(y) and plot the relationship between x1
# and the predicted mean(y) on the original scale without assuming
# normality of residuals; use the smearing estimator
# See help file for rbind.Predict for a method of showing two
# types of confidence intervals simultaneously.
set.seed(1)
x1 <- runif(300)
x2 <- runif(300)
ddist <- datadist(x1,x2)
y  <- exp(x1+x2-1+rnorm(300))
f <- ols(log(y) ~ pol(x1,2)+x2)
r <- resid(f)
smean <- function(yhat)smearingEst(yhat, exp, res, statistic='mean')
formals(smean) <- list(yhat=numeric(0), res=r[!is.na(r)])
#smean$res <- r[!is.na(r)]   # define default res argument to function
plot(Predict(f, x1, fun=smean), ylab='Predicted Mean on y-scale')

# Make an 'interaction plot', forcing the x-axis variable to be
# plotted at integer values but labeled with category levels
n <- 100
set.seed(1)
gender <- c(rep('male', n), rep('female',n))
m <- sample(c('a','b'), 2*n, TRUE)
d <-  datadist(gender, m); options(datadist='d')
anxiety <- runif(2*n) + .2*(gender=='female') + .4*(gender=='female' & m=='b')
tapply(anxiety, llist(gender,m), mean)
f <- ols(anxiety ~ gender*m)
p <- Predict(f, gender, m)
plot(p)     # horizontal dot chart; usually preferred for categorical predictors
Key(.5, .5)
plot(p, ~gender, groups='m', nlines=TRUE)
plot(p, ~m, groups='gender', nlines=TRUE)
plot(p, ~gender|m, nlines=TRUE)

options(datadist=NULL)
```

```
## Not run:
# Example in which separate curves are shown for 4 income values
# For each curve the estimated percentage of voters voting for
# the democratic party is plotted against the percent of voters
# who graduated from college.  Data are county-level percents.

incomes <- seq(22900, 32800, length=4)
# equally spaced to outer quintiles
p <- Predict(f, college, income=incomes, conf.int=FALSE)
plot(p, xlim=c(0,35), ylim=c(30,55))

# Erase end portions of each curve where there are fewer than 10 counties having
# percent of college graduates to the left of the x-coordinate being plotted,
# for the subset of counties having median family income with 1650
# of the target income for the curve

show.pts <- function(college.pts, income.pt) {
  s <- abs(income - income.pt) < 1650  #assumes income known to top frame
  x <- college[s]
  x <- sort(x[!is.na(x)])
  n <- length(x)
  low <- x[10]; high <- x[n-9]
  college.pts >= low & college.pts <= high
}

plot(p, xlim=c(0,35), ylim=c(30,55), perim=show.pts)

# Rename variables for better plotting of a long list of predictors
f <- ...
p <- Predict(f)
re <- c(trt='treatment', diabet='diabetes', sbp='systolic blood pressure')

for(n in names(re)) {
  names(p)[names(p)==n] <- re[n]
  p$.predictor.[p$.predictor.==n] <- re[n]
  }
plot(p)

## End(Not run)
```

---

plot.rexVar *plot.rexVar*

Description

Plot rexVar Result

Usage

```
## S3 method for class 'rexVar'
plot(
  x,
  xlab = "Relative Explained Variation",
  xlim = NULL,
  pch = 16,
  sort = c("descending", "ascending", "none"),
  margin = FALSE,
  height = NULL,
  width = NULL,
  ...
)
```

Arguments

x	a vector or matrix created by rexVar
xlab	x-axis label
xlim	x-axis limits; defaults to range of all values (limits and point estimates)
pch	plotting symbol for dot
sort	defaults to sorted predictors in descending order of relative explained variable. Can set to ascending or none.
margin	set to TRUE to show the REV values in the right margin if using base graphics
height	optional height in pixels for plotly graph
width	likewise optional width
...	arguments passed to dotchart2 or dotchartpl

Details

Makes a dot chart displaying the results of rexVar. Base graphics are used unless options(grType='plotly') is in effect, in which case a plotly graphic is produced with hovertext

Value

plotly graphics object if using plotly

Author(s)

Frank Harrell

plot.xmean.ordinaly *Plot Mean X vs. Ordinal Y*

Description

Separately for each predictor variable X in a formula, plots the mean of X vs. levels of Y. Then under the proportional odds assumption, the expected value of the predictor for each Y value is also plotted (as a dotted line). This plot is useful for assessing the ordinality assumption for Y separately for each X, and for assessing the proportional odds assumption in a simple univariable way. If several predictors do not distinguish adjacent categories of Y, those levels may need to be pooled. This display assumes that each predictor is linearly related to the log odds of each event in the proportional odds model. There is also an option to plot the expected means assuming a forward continuation ratio model.

Usage

```
## S3 method for class 'xmean.ordinaly'
plot(x, data, subset, na.action, subn=TRUE,
                    cr=FALSE, topcats=1, cex.points=.75, ...)
```

Arguments

`x`	an S formula. Response variable is treated as ordinal. For categorical predictors, a binary version of the variable is substituted, specifying whether or not the variable equals the modal category. Interactions or non-linear effects are not allowed.
`data`	a data frame or frame number
`subset`	vector of subscripts or logical vector describing subset of data to analyze
`na.action`	defaults to `na.keep` so all NAs are initially retained. Then NAs are deleted only for each predictor currently being plotted. Specify `na.action=na.delete` to remove observations that are missing on any of the predictors (or the response).
`subn`	set to `FALSE` to suppress a left bottom subtitle specifying the sample size used in constructing each plot
`cr`	set to `TRUE` to plot expected values by levels of the response, assuming a forward continuation ratio model holds. The function is fairly slow when this option is specified.
`topcats`	When a predictor is categorical, by default only the proportion of observations in the overall most frequent category will be plotted against response variable strata. Specify a higher value of `topcats` to make separate plots for the proportion in the `k` most frequent predictor categories, where `k` is `min(ncat-1, topcats)` and `ncat` is the number of unique values of the predictor.
`cex.points`	if `cr` is `TRUE`, specifies the size of the `"C"` that is plotted. Default is 0.75.
`...`	other arguments passed to `plot` and `lines`

Side Effects

plots

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

References

Harrell FE et al. (1998): Development of a clinical prediction model for an ordinal outcome. Stat in Med 17:909–44.

See Also

lrm, residuals.lrm, cr.setup, summary.formula, biVar.

Examples

```
# Simulate data from a population proportional odds model
set.seed(1)
n <- 400
age <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
region <- factor(sample(c('north','south','east','west'), n, replace=TRUE))
L <- .2*(age-50) + .1*(blood.pressure-120)
p12 <- plogis(L)    # Pr(Y>=1)
p2  <- plogis(L-1)  # Pr(Y=2)
p   <- cbind(1-p12, p12-p2, p2)   # individual class probabilites
# Cumulative probabilities:
cp  <- matrix(cumsum(t(p)) - rep(0:(n-1), rep(3,n)), byrow=TRUE, ncol=3)
y   <- (cp < runif(n)) %*% rep(1,3)
# Thanks to Dave Krantz <dhk@paradox.psych.columbia.edu> for this trick

par(mfrow=c(2,2))
plot.xmean.ordinaly(y ~ age + blood.pressure + region, cr=TRUE, topcats=2)
par(mfrow=c(1,1))
# Note that for unimportant predictors we don't care very much about the
# shapes of these plots.  Use the Hmisc chiSquare function to compute
# Pearson chi-square statistics to rank the variables by unadjusted
# importance without assuming any ordering of the response:
chiSquare(y ~ age + blood.pressure + region, g=3)
chiSquare(y ~ age + blood.pressure + region, g=5)
```

plotIntercepts *Plot Intercepts*

Description

Plots the step function corresponding to the intercepts in a `orm` or `lrm` model. This can be thought of as the link function of the covariate-adjusted empirical cumulative distribution function (actually 1 - ECDF). It is also related to q-q plots. For example, if a probit link function is an appropriate choice, and the residuals actually had a normal distribution (not needed by the semiparametric ordinal model), the step function of the intercepts would form a straight line.

Usage

```
plotIntercepts(fit, dots = FALSE, logt = FALSE)
```

Arguments

`fit`	an `orm` or `lrm` fit object, usually with a numeric dependent variable having many levels
`dots`	set to `TRUE` to show solid dots at the intecept values
`logt`	set to `TRUE` to use a log scale for the x-axis

Value

`ggplot2` object

Author(s)

Frank Harrell

Examples

```
## Not run:
f <- orm(y ~ x1 + x2 + x3)
plotIntercepts(f)

## End(Not run)
```

`plotp.Predict` *Plot Effects of Variables Estimated by a Regression Model Fit Using plotly*

Description

Uses `plotly` graphics (without using ggplot2) to plot the effect of one or two predictors on the linear predictor or X beta scale, or on some transformation of that scale. The first argument specifies the result of the `Predict` function. The predictor is always plotted in its original coding. Hover text shows point estimates, confidence intervals, and on the leftmost x-point, adjustment variable settings.

If `Predict` was run with no variable settings, so that each predictor is varied one at a time, the result of `plotp.Predict` is a list with two elements. The first, named `Continuous`, is a `plotly` object containing a single graphic with all the continuous predictors varying. The second, named `Categorical`, is a `plotly` object containing a single graphic with all the categorical predictors varying. If there are no categorical predictors, the value returned by by `plotp.Predict` is a single `plotly` object and not a list of objects.

If `rdata` is given, a spike histogram is drawn showing the location/density of data values for the x-axis variable. If there is a superposition variable that generated separate curves, the data density specific to each class of points is shown. The histograms are drawn by `histSpikeg`.

To plot effects instead of estimates (e.g., treatment differences as a function of interacting factors) see `contrast.rms` and `summary.rms`.

Unlike `ggplot.Predict`, `plotp.Predict` does not handle `groups`, `anova`, or `perim` arguments.

Usage

```
## S3 method for class 'Predict'
plotp(data, subset, xlim, ylim, xlab, ylab,
     rdata=NULL, nlevels=3, vnames=c('labels','names'),
     histSpike.opts=list(frac=function(f) 0.01 +
         0.02 * sqrt(f - 1)/sqrt(max(f, 2) - 1), side=1, nint=100),
     ncols=3, width=800, ...)
```

Arguments

`data`	a data frame created by `Predict`
`subset`	a subsetting expression for restricting the rows of `data` that are used in plotting. For example, predictions may have been requested for males and females but one wants to plot only females.
`xlim`	ignored unless predictors were specified to `Predict`. Specifies the x-axis limits of the single plot produced.
`ylim`	Range for plotting on response variable axis. Computed by default and includes the confidence limits.

xlab	Label for x-axis when a single plot is made, i.e., when a predictor is specified to Predict. Default is one given to asis, rcs, etc., which may have been the "label" attribute of the variable.
ylab	Label for y-axis. If fun is not given, default is "log Odds" for lrm, "log Relative Hazard" for cph, name of the response variable for ols, TRUE or log(TRUE) for psm, or "X * Beta" otherwise. Specify ylab=NULL to omit y-axis labels.
rdata	a data frame containing the original raw data on which the regression model were based, or at least containing the x-axis and grouping variable. If rdata is present and contains the needed variables, the original data are added to the graph in the form of a spike histogram using histSpikeg in the Hmisc package.
nlevels	A non-numeric x-axis variable with nlevels or fewer unique values will cause a horizontal dot plot to be drawn instead of an x-y plot.
vnames	applies to the case where multiple plots are produced separately by predictor. Set to 'names' to use variable names instead of labels for these small plots.
histSpike.opts	a list containing named elements that specifies parameters to histSpikeg when rdata is given. The col parameter is usually derived from other plotting information and not specified by the user.
ncols	number of columns of plots to use when plotting multiple continuous predictors
width	width in pixels for plotly graphics
...	ignored

Value

a plotly object or a list containing two elements, each one a plotly object

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Fox J, Hong J (2009): Effect displays in R for multinomial and proportional-odds logit models: Extensions to the effects package. J Stat Software 32 No. 1.

See Also

Predict, rbind.Predict, datadist, predictrms, contrast.rms, summary.rms, rms, rmsMisc, plot.Predict, ggplot.Predict, histSpikeg, Overview

Examples

```
## Not run:
n <- 350     # define sample size
set.seed(17) # so can reproduce the results
```

```
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
label(age)            <- 'Age'      # label is in Hmisc
label(cholesterol)    <- 'Total Cholesterol'
label(blood.pressure) <- 'Systolic Blood Pressure'
label(sex)            <- 'Sex'
units(cholesterol)    <- 'mg/dl'   # uses units.default in Hmisc
units(blood.pressure) <- 'mmHg'

# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
    (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male')) +
    .01 * (blood.pressure - 120)
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)

ddist <- datadist(age, blood.pressure, cholesterol, sex)
options(datadist='ddist')

fit <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)),
           x=TRUE, y=TRUE)

p <- plotp(Predict(fit))
p$Continuous
p$Categorical
# When using Rmarkdown html notebook, best to use
# prList(p) to render the two objects
plotp(Predict(fit), rdata=llist(blood.pressure, age))$Continuous
# spike histogram plot for two of the predictors

p <- Predict(fit, name=c('age','cholesterol'))   # Make 2 plots
plotp(p)

p <- Predict(fit, age, sex)
plotp(p, rdata=llist(age,sex))
# rdata= allows rug plots (1-dimensional scatterplots)
# on each sex's curve, with sex-
# specific density of age
# If data were in data frame could have used that
p <- Predict(fit, age=seq(20,80,length=100), sex='male', fun=plogis)
# works if datadist not used
plotp(p, ylab='P')
# plot predicted probability in place of log odds

# Compute predictions for three predictors, with superpositioning or
# conditioning on sex, combined into one graph

p1 <- Predict(fit, age, sex)
p2 <- Predict(fit, cholesterol, sex)
p3 <- Predict(fit, blood.pressure, sex)
p <- rbind(age=p1, cholesterol=p2, blood.pressure=p3)
```

```
plotp(p, ncols=2, rdata=llist(age, cholesterol, sex))

## End(Not run)
```

---

poma — *Examine proportional odds and parallelism assumptions of ‘orm‘ and ‘lrm‘ model fits.*

Description

Based on codes and strategies from Frank Harrell’s canonical ‘Regression Modeling Strategies‘ text

Usage

```
poma(mod.orm, cutval, minfreq = 15, ...)
```

Arguments

`mod.orm`	Model fit of class ‘orm‘ or ‘lrm‘. For ‘fit.mult.impute‘ objects, ‘poma‘ will refit model on a singly-imputed data-set
`cutval`	Numeric vector; sequence of observed values to cut outcome
`minfreq`	Numeric vector; an ‘impactPO‘ argument which specifies the minimum sample size to allow for the least frequent category of the dependent variable.
`...`	parameters to pass to ‘impactPO‘ function such as ‘newdata‘, ‘nonpo‘, and ‘B‘.

Details

Strategy 1: Compare PO model fit with models that relax the PO assumption (for discrete response variable)
Strategy 2: Apply different link functions to Prob of Binary Ys (defined by cutval). Regress transformed outcome on combined X and assess constancy of slopes (betas) across cut-points
Strategy 3: Generate score residual plot for each predictor (for response variable with <10 unique levels)
Strategy 4: Assess parallelism of link function transformed inverse CDFs curves for different XBeta levels (for response variables with >=10 unique levels)

Author(s)

Yong Hao Pua <puayonghao@gmail.com>

See Also

Harrell FE. *Regression Modeling Strategies: with applications to linear models, logistic and ordinal regression, and survival analysis.* New York: Springer Science, LLC, 2015.
Harrell FE. Statistical Thinking - Assessing the Proportional Odds Assumption and Its Impact. https://www.fharrell.com/post/impactpo/. Published March 9, 2022. Accessed January 13, 2023. [rms::impactPO()]

Examples

```
## Not run:
## orm model (response variable has fewer than 10 unique levels)
mod.orm <- orm(carb ~ cyl + hp , x = TRUE, y = TRUE, data = mtcars)
poma(mod.orm)


## runs rms::impactPO when its args are supplied
## More examples: (https://yhpua.github.io/poma/)
d <- expand.grid(hp = c(90, 180), vs = c(0, 1))
mod.orm <- orm(cyl ~ vs + hp , x = TRUE, y = TRUE, data = mtcars)
poma(mod.orm, newdata = d)


## orm model (response variable has >=10 unique levels)
mod.orm <- orm(mpg ~ cyl + hp , x=TRUE, y=TRUE, data = mtcars)
poma(mod.orm)


## orm model using imputation
dat <- mtcars
## introduce NAs
dat[sample(rownames(dat), 10), "cyl"] <- NA
im <- aregImpute(~ cyl + wt + mpg + am, data = dat)
aa <- fit.mult.impute(mpg ~ cyl + wt , xtrans = im, data = dat, fitter = orm)
poma(aa)

## End(Not run)
```

---

pphsm *Parametric Proportional Hazards form of AFT Models*

Description

Translates an accelerated failure time (AFT) model fitted by psm to proportional hazards form, if the fitted model was a Weibull or exponential model (extreme value distribution with "log" link).

Usage

```
pphsm(fit)
## S3 method for class 'pphsm'
print(x, digits=max(options()$digits - 4, 3),
correlation=TRUE, ...)
## S3 method for class 'pphsm'
vcov(object, ...)
```

Arguments

fit	fit object created by psm
x	result of psm
digits	how many significant digits are to be used for the returned value
correlation	set to FALSE to suppress printing of correlation matrix of parameter estimates
...	ignored
object	a pphsm object

Value

a new fit object with transformed parameter estimates

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

psm, summary.rms, print.pphsm

Examples

```
require(survival)
set.seed(1)
S <- Surv(runif(100))
x <- runif(100)
dd <- datadist(x); options(datadist='dd')
f <- psm(S ~ x, dist="exponential")
summary(f)        # effects on log(T) scale
f.ph <- pphsm(f)
## Not run: summary(f.ph)     # effects on hazard ratio scale
options(datadist=NULL)
```

---

predab.resample *Predictive Ability using Resampling*

Description

predab.resample is a general-purpose function that is used by functions for specific models. It computes estimates of optimism of, and bias-corrected estimates of a vector of indexes of predictive accuracy, for a model with a specified design matrix, with or without fast backward step-down of predictors. If bw=TRUE, the design matrix x must have been created by ols, lrm, or cph. If bw=TRUE,

predab.resample stores as the kept attribute a logical matrix encoding which factors were selected at each repetition.

Unless conf.int is FALSE or 0, the function computes approximate bootstrap confidence intervals for overfitting-corrected predictive performance measures using the method of Harrell (2025) described and tested at https://www.fharrell.com/post/bootcal/ and inspired by Noma et al (2021).

Usage

```
predab.resample(fit.orig, fit, measure,
                method=c("boot","crossvalidation",".632","randomization"),
                bw=FALSE, B=50, conf.int=0.95, pr=FALSE, prmodsel=TRUE,
                rule="aic", type="residual", sls=.05, aics=0,
                tol=.Machine$double.eps, force=NULL, estimates=TRUE,
                non.slopes.in.x=TRUE, kint=1,
                cluster, subset, group=NULL,
               allow.varying.intercepts=FALSE, debug=FALSE, saveraw=FALSE, ...)
```

Arguments

fit.orig — object containing the original full-sample fit, with the x=TRUE and y=TRUE options specified to the model fitting function. This model should be the FULL model including all candidate variables ever excluded because of poor associations with the response.

fit — a function to fit the model, either the original model fit, or a fit in a sample. fit has as arguments x,y, iter, penalty, penalty.matrix, xcol, and other arguments passed to predab.resample. If you don't want iter as an argument inside the definition of fit, add ... to the end of its argument list. iter is passed to fit to inform the function of the sampling repetition number (0=original sample). If bw=TRUE, fit should allow for the possibility of selecting no predictors, i.e., it should fit an intercept-only model if the model has intercept(s). fit must return objects coef and fail (fail=TRUE if fit failed due to singularity or non-convergence - these cases are excluded from summary statistics). fit must add design attributes to the returned object if bw=TRUE. The penalty.matrix parameter is not used if penalty=0. The xcol vector is a vector of columns of X to be used in the current model fit. For ols and psm it includes a 1 for the intercept position. xcol is not defined if iter=0 unless the initial fit had been from a backward step-down. xcol is used to select the correct rows and columns of penalty.matrix for the current variables selected, for example.

measure — a function to compute a vector of indexes of predictive accuracy for a given fit. For method=".632" or method="crossval", it will make the most sense for measure to compute only indexes that are independent of sample size. The measure function should take the following arguments or use ...: xbeta (X beta for current fit), y, evalfit, fit, iter, and fit.orig. iter is as in fit. evalfit is set to TRUE by predab.resample if the fit is being evaluated on the sample used to make the fit, FALSE otherwise; fit.orig is the fit object returned by the original fit on the whole sample. Using evalfit will sometimes save computations. For example, in bootstrapping the area under an ROC curve

for a logistic regression model, `lrm` already computes the area if the fit is on the training sample. `fit.orig` is used to pass computed configuration parameters from the original fit such as quantiles of predicted probabilities that are used as cut points in other samples. The vector created by `measure` should have `names()` associated with it if the model performance measures are indexes such as the Brier score or calibration slope. The vector must not have names if it corresponds to a nonparametric calibration curve.

`method` The default is `"boot"` for ordinary bootstrapping (Efron, 1983, Eq. 2.10). Use `".632"` for Efron's `.632` method (Efron, 1983, Section 6 and Eq. 6.10), `"crossvalidation"` for grouped cross–validation, `"randomization"` for the randomization method. May be abbreviated down to any level, e.g. `"b"`, `"."`, `"cross"`, `"rand"`.

`bw` Set to `TRUE` to do fast backward step-down for each training sample. Default is `FALSE`.

`B` Number of repetitions, default=50, which is far too small. For `method="crossvalidation"`, this is also the number of groups the original sample is split into.

`conf.int` confidence level for approximate confidence limits for overfitting-corrected indexes. Set to `FALSE` or 0 to not compute limits. For calibration,

`pr` `TRUE` to print results for each sample. Default is `FALSE`. Also controls printing of number of divergent or singular samples.

`prmodsel` set to `FALSE` to suppress printing of model selection output such as that from `fastbw`.

`rule` Stopping rule for fastbw, `"aic"` or `"p"`. Default is `"aic"` to use Akaike's information criterion.

`type` Type of statistic to use in stopping rule for fastbw, `"residual"` (the default) or `"individual"`.

`sls` Significance level for stopping in fastbw if `rule="p"`. Default is `.05`.

`aics` Stopping criteria for `rule="aic"`. Stops deleting factors when chi-square - 2 times d.f. falls below `aics`. Default is `0`.

`tol` Tolerance for singularity checking. Is passed to `fit` and `fastbw`.

`force` see `fastbw`

`estimates` see `print.fastbw`

`non.slopes.in.x` set to `FALSE` if the design matrix `x` does not have columns for intercepts and these columns are needed

`kint` For multiple intercept models such as the ordinal logistic model, you may specify which intercept to use as `kint`. This affects the linear predictor that is passed to `measure`.

`cluster` Vector containing cluster identifiers. This can be specified only if `method="boot"`. If it is present, the bootstrap is done using sampling with replacement from the clusters rather than from the original records. If this vector is not the same length as the number of rows in the data matrix used in the fit, an attempt will be made to use `naresid` on `fit.orig` to conform `cluster` to the data. See `bootcov` for more about this.

subset — specify a vector of positive or negative integers or a logical vector when you want to have the `measure` function compute measures of accuracy on a subset of the data. The whole dataset is still used for all model development. For example, you may want to `validate` or `calibrate` a model by assessing the predictions on females when the fit was based on males and females. When you use `cr.setup` to build extra observations for fitting the continuation ratio ordinal logistic model, you can use `subset` to specify which `cohort` or observations to use for deriving indexes of predictive accuracy. For example, specify `subset=cohort=="all"` to validate the model for the first layer of the continuation ratio model (Prob(Y=0)).

group — a grouping variable used to stratify the sample upon bootstrapping. This allows one to handle k-sample problems, i.e., each bootstrap sample will be forced to selected the same number of observations from each level of group as the number appearing in the original dataset.

allow.varying.intercepts — set to `TRUE` to not throw an error if the number of intercepts varies from fit to fit

debug — set to `TRUE` to print subscripts of all training and test samples

saveraw — set to `TRUE` to store a list named `.predab_raw.` in the global environment. The list has the elements `orig` (original estimates of performance indexes), `btrain` (a matrix with up to `B` bootstrap repetitions of indexes computed on training samples), and `btest` (a similar matrix but computing indexes on the test samples). For bootstrapping, training samples are bootstrap samples and test samples are the original data.

... — The user may add other arguments here that are passed to `fit` and `measure`.

Details

For `method=".632"`, the program stops with an error if every observation is not omitted at least once from a bootstrap sample. Efron's ".632" method was developed for measures that are formulated in terms on per-observation contributions. In general, error measures (e.g., ROC areas) cannot be written in this way, so this function uses a heuristic extension to Efron's formulation in which it is assumed that the average error measure omitting the `i`th observation is the same as the average error measure omitting any other observation. Then weights are derived for each bootstrap repetition and weighted averages over the `B` repetitions can easily be computed.

Value

a matrix of class `"validate"` with rows corresponding to indexes computed by `measure`, and the following columns:

index.orig — indexes in original overall fit

training — average indexes in training samples

test — average indexes in test samples

optimism — average `training-test` except for `method=".632"` - is .632 times (`index.orig - test`)

index.corrected — `index.orig-optimism`

n number of successful repetitions with the given index non-missing

. Also contains an attribute keepinfo if measure returned such an attribute when run on the original fit.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Efron B, Tibshirani R (1997). Improvements on cross-validation: The .632+ bootstrap method. JASA 92:548–560.

Noma H, et al (2021). Confidence intervals of prediction accuracy measures for multivariable prediction models based on the bootstrap-based optimism correction methods. Statistics in Medicine 40:5691-5701.

Harrell FE (2025). Bootstrap confidence limits for bootstrap overfitting-corrected model performance, https://www.fharrell.com/post/bootcal/

See Also

rms, validate, fastbw, lrm, ols, cph, bootcov, setPb

Examples

```
# See the code for validate.ols for an example of the use of
# predab.resample
```

Predict *Compute Predicted Values and Confidence Limits*

Description

Predict allows the user to easily specify which predictors are to vary. When the vector of values over which a predictor should vary is not specified, the range will be all levels of a categorical predictor or equally-spaced points between the datadist "Low:prediction" and "High:prediction" values for the variable (datadist by default uses the 10th smallest and 10th largest predictor values in the dataset). Predicted values are the linear predictor (X beta), a user-specified transformation of that scale, or estimated probability of surviving past a fixed single time point given the linear predictor. Predict is usually used for plotting predicted values but there is also a print method.

When the first argument to Predict is a fit object created by bootcov with coef.reps=TRUE, confidence limits come from the stored matrix of bootstrap repetitions of coefficients, using bootstrap percentile nonparametric confidence limits, basic bootstrap, or BCa limits. Such confidence intervals do not make distributional assumptions. You can force Predict to instead use the bootstrap

covariance matrix by setting usebootcoef=FALSE. If coef.reps was FALSE, usebootcoef=FALSE is the default.

There are ggplot, plotp, and plot methods for Predict objects that makes it easy to show predicted values and confidence bands.

The rbind method for Predict objects allows you to create separate sets of predictions under different situations and to combine them into one set for feeding to plot.Predict, ggplot.Predict, or plotp.Predict. For example you might want to plot confidence intervals for means and for individuals using ols, and have the two types of confidence bands be superposed onto one plot or placed into two panels. Another use for rbind is to combine predictions from quantile regression models that predicted three different quantiles.

If conf.type="simultaneous", simultaneous (over all requested predictions) confidence limits are computed. See the predictrms function for details.

If fun is given, conf.int > 0, the model is not a Bayesian model, and the bootstrap was not used, fun may return limits attribute when fun computed its own confidence limits. These confidence limits will be functions of the design matrix, not just the linear predictor.

Usage

```
Predict(object, ..., fun=NULL, funint=TRUE,
        type = c("predictions", "model.frame", "x"),
        np = 200, conf.int = 0.95,
        conf.type = c("mean", "individual","simultaneous"),
        usebootcoef=TRUE, boot.type=c("percentile", "bca", "basic"),
        posterior.summary=c('mean', 'median', 'mode'),
        adj.zero = FALSE, ref.zero = FALSE,
        kint=NULL, ycut=NULL, time = NULL, loglog = FALSE, digits=4, name,
        factors=NULL, offset=NULL)

## S3 method for class 'Predict'
print(x, ...)

## S3 method for class 'Predict'
rbind(..., rename)
```

Arguments

object — an rms fit object, or for print the result of Predict. options(datadist="d") must have been specified (where d was created by datadist), or it must have been in effect when the the model was fitted.

... — One or more variables to vary, or single-valued adjustment values. Specify a variable name without an equal sign to use the default display range, or any range you choose (e.g. seq(0,100,by=2),c(2,3,7,14)). The default list of values for which predictions are made is taken as the list of unique values of the variable if they number fewer than 11. For variables with > 10 unique values, np equally spaced values in the range are used for plotting if the range is not specified. Variables not specified are set to the default adjustment value limits[2], i.e. the median for continuous variables and a reference category for

non-continuous ones. Later variables define adjustment settings. For categorical variables, specify the class labels in quotes when specifying variable values. If the levels of a categorical variable are numeric, you may omit the quotes. For variables not described using `datadist`, you must specify explicit ranges and adjustment settings for predictors that were in the model. If no variables are specified in ..., predictions will be made by separately varying all predictors in the model over their default range, holding the other predictors at their adjustment values. This has the same effect as specifying `name` as a vector containing all the predictors. For `rbind`, ... represents a series of results from `Predict`. If you name the results, these names will be taken as the values of the new `.set.` variable added to the concatenated data frames. See an example below.

`fun` an optional transformation of the linear predictor. Specify `fun='mean'` if the fit is a proportional odds model fit and you ran `bootcov` with `coef.reps=TRUE`. This will let the mean function be re-estimated for each bootstrap rep to properly account for all sources of uncertainty in estimating the mean response. `fun` can be a general function and can compute confidence limits (stored as a list in the `limits` attribute) of the transformed parameters such as means.

`funint` set to `FALSE` if `fun` is not a function such as the result of `Mean`, `Quantile`, or `ExProb` that contains an `intercepts` argument

`type` defaults to providing predictions. Set to `"model.frame"` to return a data frame of predictor settings used. Set to `"x"` to return the corresponding design matrix constructed from the predictor settings.

`np` the number of equally-spaced points computed for continuous predictors that vary, i.e., when the specified value is omitted (with the variable name appearing without an equals sign) or is `NA`

`conf.int` confidence level (highest posterior density interval probability for Bayesian models). Default is 0.95. Specify `FALSE` to suppress.

`conf.type` type of confidence interval. Default is `"mean"` which applies to all models. For models containing a residual variance (e.g, `ols`), you can specify `conf.type="individual"` instead, to obtain limits on the predicted value for an individual subject. Specify `conf.type="simultaneous"` to obtain simultaneous confidence bands for mean predictions with family-wise coverage of `conf.int`.

`usebootcoef` set to `FALSE` to force the use of the bootstrap covariance matrix estimator even when bootstrap coefficient reps are present

`boot.type` set to `'bca'` to compute BCa confidence limits or `'basic'` to use the basic bootstrap. The default is to compute percentile intervals

`posterior.summary`
defaults to using the posterior mean of the regression coefficients. Specify `'mode'` or `'median'` to instead use the other summaries.

`adj.zero` Set to `TRUE` to adjust all non-plotted variables to 0 (or reference cell for categorical variables) and to omit intercept(s) from consideration. Default is `FALSE`.

`ref.zero` Set to `TRUE` to subtract a constant from $X\beta$ before plotting so that the reference value of the `x`-variable yields `y=0`. This is done before applying function `fun`. This is especially useful for Cox models to make the hazard ratio be 1.0 at reference values, and the confidence interval have width zero. To set the reference

value, either (a) set the reference value by editing the `datadist` object prior to fitting the model, or (b) if the model is already fit, edit the `datadist` object and then run the update command.

`kint` This is only useful in a multiple intercept model such as the ordinal logistic model. There to use to second of three intercepts, for example, specify `kint=2`. The default is 1 for `lrm` and the middle intercept corresponding to the median `y` for `orm` or `blrm`. You can specify `ycut` instead, and the intercept corresponding to Y >= ycut will be used for `kint`.

`ycut` for an ordinal model specifies the Y cutoff to use in evaluating departures from proportional odds, when the constrained partial proportional odds model is used. When omitted, `ycut` is implied by `kint`. The only time it is absolutely mandatory to specify `ycut` is when computed an effect (e.g., odds ratio) at a level of the response variable that did not occur in the data. This would only occur when the `cppo` function given to `blrm` is a continuous function.

`time` Specify a single time `u` to cause function `survest` to be invoked to plot the probability of surviving until time `u` when the fit is from `cph` or `psm`.

`loglog` Specify `loglog=TRUE` to plot `log[-log(survival)]` instead of survival, when `time` is given.

`digits` Controls how "adjust-to" values are plotted. The default is 4 significant digits.

`name` Instead of specifying the variables to vary in the `variables` (`...`) list, you can specify one or more variables by specifying a vector of character string variable names in the `name` argument. Using this mode you cannot specify a list of variable values to use; prediction is done as if you had said e.g. `age` without the equal sign. Also, interacting factors can only be set to their reference values using this notation.

`factors` an alternate way of specifying `...`, mainly for use by `survplot` or `gendata`. This must be a list with one or more values for each variable listed, with `NA` values for default ranges.

`offset` a list containing one value for one variable, which is mandatory if the model included an offset term. The variable name must match the innermost variable name in the offset term. The single offset is added to all predicted values.

`x` an object created by `Predict`

`rename` If you are concatenating predictor sets using `rbind` and one or more of the variables were renamed for one or more of the sets, but these new names represent different versions of the same predictors (e.g., using or not using imputation), you can specify a named character vector to rename predictors to a central name. For example, specify `rename=c(age.imputed='age', corrected.bp='bp')` to rename from old names `age.imputed, corrected.bp` to `age, bp`. This happens before concatenation of rows.

Details

When there are no intercepts in the fitted model, plot subtracts adjustment values from each factor while computing variances for confidence limits.

Specifying `time` will not work for Cox models with time-dependent covariables. Use `survest` or `survfit` for that purpose.

Value

a data frame containing all model predictors and the computed values `yhat`, `lower`, `upper`, the latter two if confidence intervals were requested. The data frame has an additional `class "Predict"`. If `name` is specified or no predictors are specified in ..., the resulting data frame has an additional variable called `.predictor.` specifying which predictor is currently being varied. `.predictor.` is handy for use as a paneling variable in `lattice` or `ggplot2` graphics.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

`plot.Predict`, `ggplot.Predict`, `plotp.Predict`, `datadist`, `predictrms`, `contrast.rms`, `summary.rms`, `rms`, `rms.trans`, `survest`, `survplot`, `rmsMisc`, `transace`, `rbind`, `bootcov`, `bootBCa`, `boot.ci`

Examples

```
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
label(age)            <- 'Age'      # label is in Hmisc
label(cholesterol)    <- 'Total Cholesterol'
label(blood.pressure) <- 'Systolic Blood Pressure'
label(sex)            <- 'Sex'
units(cholesterol)    <- 'mg/dl'   # uses units.default in Hmisc
units(blood.pressure) <- 'mmHg'

# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
  (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male'))
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)

ddist <- datadist(age, blood.pressure, cholesterol, sex)
options(datadist='ddist')

fit <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)))
Predict(fit, age, cholesterol, np=4)
Predict(fit, age=seq(20,80,by=10), sex, conf.int=FALSE)
Predict(fit, age=seq(20,80,by=10), sex='male')  # works if datadist not used
# Get simultaneous confidence limits accounting for making 7 estimates
# Predict(fit, age=seq(20,80,by=10), sex='male', conf.type='simult')
# (this needs the multcomp package)

ddist$limits$age[2] <- 30    # make 30 the reference value for age
```

```
# Could also do: ddist$limits["Adjust to","age"] <- 30
fit <- update(fit)   # make new reference value take effect
Predict(fit, age, ref.zero=TRUE, fun=exp)

# Make two curves, and plot the predicted curves as two trellis panels
w <- Predict(fit, age, sex)
require(lattice)
xyplot(yhat ~ age | sex, data=w, type='l')
# To add confidence bands we need to use the Hmisc xYplot function in
# place of xyplot
xYplot(Cbind(yhat,lower,upper) ~ age | sex, data=w,
       method='filled bands', type='l', col.fill=gray(.95))
# If non-displayed variables were in the model, add a subtitle to show
# their settings using title(sub=paste('Adjusted to',attr(w,'info')$adjust),adj=0)
# Easier: feed w into plot.Predict, ggplot.Predict, plotp.Predict
## Not run:
# Predictions form a parametric survival model
require(survival)
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('Male','Female'), n,
              rep=TRUE, prob=c(.6, .4)))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
t <- -log(runif(n))/h
label(t) <- 'Follow-up Time'
e <- ifelse(t<=cens,1,0)
t <- pmin(t, cens)
units(t) <- "Year"
ddist <- datadist(age, sex)
Srv <- Surv(t,e)

# Fit log-normal survival model and plot median survival time vs. age
f <- psm(Srv ~ rcs(age), dist='lognormal')
med <- Quantile(f)       # Creates function to compute quantiles
                         # (median by default)
Predict(f, age, fun=function(x)med(lp=x))
# Note: This works because med() expects the linear predictor (X*beta)
#       as an argument.  Would not work if use
#       ref.zero=TRUE or adj.zero=TRUE.
# Also, confidence intervals from this method are approximate since
# they don't take into account estimation of scale parameter

# Fit an ols model to log(y) and plot the relationship between x1
# and the predicted mean(y) on the original scale without assuming
# normality of residuals; use the smearing estimator.  Before doing
# that, show confidence intervals for mean and individual log(y),
# and for the latter, also show bootstrap percentile nonparametric
# pointwise confidence limits
set.seed(1)
x1 <- runif(300)
```

```
x2 <- runif(300)
ddist <- datadist(x1,x2); options(datadist='ddist')
y  <- exp(x1+ x2 - 1 + rnorm(300))
f  <- ols(log(y) ~ pol(x1,2) + x2, x=TRUE, y=TRUE)  # x y for bootcov
fb <- bootcov(f, B=100)
pb <- Predict(fb, x1, x2=c(.25,.75))
p1 <- Predict(f,  x1, x2=c(.25,.75))
p <- rbind(normal=p1, boot=pb)
plot(p)

p1 <- Predict(f, x1, conf.type='mean')
p2 <- Predict(f, x1, conf.type='individual')
p  <- rbind(mean=p1, individual=p2)
plot(p, label.curve=FALSE)   # uses superposition
plot(p, ~x1 | .set.)         # 2 panels

r <- resid(f)
smean <- function(yhat)smearingEst(yhat, exp, res, statistic='mean')
formals(smean) <- list(yhat=numeric(0), res=r[!is.na(r)])
#smean$res <- r[!is.na(r)]   # define default res argument to function
Predict(f, x1, fun=smean)

## Example using offset
g <- Glm(Y ~ offset(log(N)) + x1 + x2, family=poisson)
Predict(g, offset=list(N=100))

## End(Not run)
options(datadist=NULL)
```

`predict.lrm` *Predicted Values for Binary and Ordinal Logistic Models*

Description

Computes a variety of types of predicted values for fits from `lrm` and `orm`, either from the original dataset or for new observations. The `Mean.lrm` and `Mean.orm` functions produce an R function to compute the predicted mean of a numeric ordered response variable given the linear predictor, which is assumed to use the first intercept when it was computed. The returned function has two optional arguments if confidence intervals are desired: `conf.int` and the design matrix `X`. When this derived function is called with nonzero `conf.int`, an attribute named `limits` is attached to the estimated mean. This is a list with elements `lower` and `upper` containing normal approximations for confidence limits using the delta method.

For `orm` fits on censored data, the function created by `Mean.orm` has an argument `tmax` which specifies the restriction time for mean restricted survival time.

Usage

```
## S3 method for class 'lrm'
predict(object, ..., type=c("lp", "fitted",
```

```
            "fitted.ind", "mean", "x", "data.frame",
            "terms", "cterms", "ccterms", "adjto","adjto.data.frame",
            "model.frame"), se.fit=FALSE, codes=FALSE)
## S3 method for class 'orm'
predict(object, ..., type=c("lp", "fitted",
            "fitted.ind", "mean", "x", "data.frame",
            "terms", "cterms", "ccterms", "adjto","adjto.data.frame",
            "model.frame"), se.fit=FALSE, codes=FALSE)

## S3 method for class 'lrm'
Mean(object, codes=FALSE, ...)
## S3 method for class 'orm'
Mean(object, codes=FALSE, ...)
```

Arguments

`object`	a object created by `lrm` or `orm`
`...`	arguments passed to `predictrms`, such as `kint` and `newdata` (which is used if you are predicting `out of data`). See `predictrms` to see how NAs are handled. Ignored for other functions.
`type`	See `predict.rms` for `"x"`, `"data.frame"`, `"terms"`, `"cterms"`, `"ccterms"`, `"adjto"`, `"adjto.data.frame"` and `"model.frame"`. `type="lp"` is used to get linear predictors (using the first intercept by default; specify `kint` to use others). `type="fitted"` is used to get all the probabilities $Y \geq j$. `type="fitted.ind"` gets all the individual probabilities $Y = j$ (not recommended for `orm` fits). For an ordinal response variable, `type="mean"` computes the estimated mean Y by summing values of Y multiplied by the estimated $Prob(Y = j)$. If Y was a character or `factor` object, the levels are the character values or factor levels, so these must be translatable to numeric, unless `codes=TRUE`. See the Hannah and Quigley reference below for the method of estimating (and presenting) the mean score. If you specify `type="fitted","fitted.ind","mean"` you may not specify `kint`.
`se.fit`	applies only to `type="lp"`, to get standard errors.
`codes`	if `TRUE`, `type="mean"`, `Mean.lrm`, and `Mean.orm` use the integer codes $1, 2, \ldots, k$ for the k-level response in computing the predicted mean response.

Value

a vector (`type="lp"` with `se.fit=FALSE`, or `type="mean"` or only one observation being predicted), a list (with elements `linear.predictors` and `se.fit` if `se.fit=TRUE`), a matrix (`type="fitted"` or `type="fitted.ind"`), a data frame, or a design matrix. For `Mean.lrm` and `Mean.orm`, the result is an R function.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University

fh@fharrell.com
For the Quantile function:
Qi Liu and Shengxin Tu
Department of Biostatistics, Vanderbilt University

References

Hannah M, Quigley P: Presentation of ordinal regression analysis on the original scale. Biometrics 52:771–5; 1996.

See Also

lrm, orm, predict.rms, naresid, contrast.rms

Examples

```
# See help for predict.rms for several binary logistic
# regression examples


# Examples of predictions from ordinal models
set.seed(1)
y <- factor(sample(1:3, 400, TRUE), 1:3, c('good','better','best'))
x1 <- runif(400)
x2 <- runif(400)
f <- lrm(y ~ rcs(x1,4)*x2, x=TRUE)     #x=TRUE needed for se.fit
# Get 0.95 confidence limits for Prob[better or best]
L <- predict(f, se.fit=TRUE)           #omitted kint= so use 1st intercept
plogis(with(L, linear.predictors + 1.96*cbind(-se.fit,se.fit)))
predict(f, type="fitted.ind")[1:10,]   #gets Prob(better) and all others
d <- data.frame(x1=c(.1,.5),x2=c(.5,.15))
predict(f, d, type="fitted")        # Prob(Y>=j) for new observation
predict(f, d, type="fitted.ind")    # Prob(Y=j)
predict(f, d, type='mean', codes=TRUE) # predicts mean(y) using codes 1,2,3
m <- Mean(f, codes=TRUE)
lp <- predict(f, d)
m(lp)
# Can use function m as an argument to Predict or nomogram to
# get predicted means instead of log odds or probabilities
dd <- datadist(x1,x2); options(datadist='dd')
m
plot(Predict(f, x1, fun=m), ylab='Predicted Mean')
# Note: Run f through bootcov with coef.reps=TRUE to get proper confidence
# limits for predicted means from the prop. odds model
options(datadist=NULL)
```

predictrms *Predicted Values from Model Fit*

Description

The `predict` function is used to obtain a variety of values or predicted values from either the data used to fit the model (if `type="adjto"` or `"adjto.data.frame"` or if `x=TRUE` or `linear.predictors=TRUE` were specified to the modeling function), or from a new dataset. Parameters such as knots and factor levels used in creating the design matrix in the original fit are "remembered". See the `Function` function for another method for computing the linear predictors. `predictrms` is an internal utility function that is for the other functions.

Usage

```
predictrms(fit, newdata=NULL,
           type=c("lp", "x", "data.frame", "terms", "cterms", "ccterms",
             "adjto", "adjto.data.frame", "model.frame"),
           se.fit=FALSE, conf.int=FALSE,
           conf.type=c('mean', 'individual', 'simultaneous'),
           kint=NULL, na.action=na.keep, expand.na=TRUE,
           center.terms=type=="terms", ref.zero=FALSE,
           posterior.summary=c('mean', 'median', 'mode'),
           second=FALSE, ...)
## S3 method for class 'bj'
predict(object, newdata,
        type=c("lp", "x", "data.frame", "terms", "cterms", "ccterms",
               "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE,
        conf.type=c('mean','individual','simultaneous'),
        kint=1,
        na.action=na.keep, expand.na=TRUE,
        center.terms=type=="terms", ...) # for bj

## S3 method for class 'cph'
predict(object, newdata=NULL,
        type=c("lp", "x", "data.frame", "terms", "cterms", "ccterms",
               "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE,
        conf.type=c('mean','individual','simultaneous'),
        kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=type=="terms", ...) # cph

## S3 method for class 'Glm'
predict(object, newdata,
        type= c("lp", "x", "data.frame", "terms", "cterms", "ccterms",
                "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE,
```

```
        conf.type=c('mean','individual','simultaneous'),
        kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=type=="terms", ...) # Glm

## S3 method for class 'Gls'
predict(object, newdata,
        type=c("lp", "x", "data.frame", "terms", "cterms", "ccterms",
               "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE,
        conf.type=c('mean','individual','simultaneous'),
        kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=type=="terms", ...) # Gls

## S3 method for class 'ols'
predict(object, newdata,
        type=c("lp", "x", "data.frame", "terms", "cterms", "ccterms",
               "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE,
        conf.type=c('mean','individual','simultaneous'),
        kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=type=="terms", ...) # ols

## S3 method for class 'psm'
predict(object, newdata,
        type=c("lp", "x", "data.frame", "terms", "cterms", "ccterms",
               "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE,
        conf.type=c('mean','individual','simultaneous'),
        kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=type=="terms", ...) # psm
```

Arguments

`object`, `fit` a fit object with an `rms` fitting function

`newdata` An S data frame, list or a matrix specifying new data for which predictions are desired. If `newdata` is a list, it is converted to a matrix first. A matrix is converted to a data frame. For the matrix form, categorical variables (`catg` or `strat`) must be coded as integer category numbers corresponding to the order in which value labels were stored. For list or matrix forms, `matrx` factors must be given a single value. If this single value is the S missing value `NA`, the adjustment values of matrx (the column medians) will later replace this value. If the single value is not `NA`, it is propagated throughout the columns of the `matrx` factor. For `factor` variables having numeric levels, you can specify the numeric values in `newdata` without first converting the variables to factors. These numeric values are checked to make sure they match a level, then the variable is converted internally to a `factor`. It is most typical to use a data frame for newdata, and the S function `expand.grid` is very handy here. For example, one may specify `newdata=expand.grid(age=c(10,20,30),`

```
race=c("black","white","other"),
chol=seq(100,300,by=25)).
```

type — Type of output desired. The default is "lp" to get the linear predictors - predicted $X\beta$. For Cox models, these predictions are centered. You may specify "x" to get an expanded design matrix at the desired combinations of values, "data.frame" to get an S data frame of the combinations, "model.frame" to get a data frame of the transformed predictors, "terms" to get a matrix with each column being the linear combination of variables making up a factor (with separate terms for interactions), "cterms" ("combined terms") to not create separate terms for interactions but to add all interaction terms involving each predictor to the main terms for each predictor, "ccterms" to combine all related terms (related through interactions) and their interactions into a single column, "adjto" to return a vector of limits[2] (see datadist) in coded form, and "adjto.data.frame" to return a data frame version of these central adjustment values. Use of type="cterms" does not make sense for a strat variable that does not interact with another variable. If newdata is not given, predict will attempt to return information stored with the fit object if the appropriate options were used with the modeling function (e.g., x, y, linear.predictors, se.fit).

se.fit — Defaults to FALSE. If type="linear.predictors", set se.fit=TRUE to return a list with components linear.predictors and se.fit instead of just a vector of fitted values. For Cox model fits, standard errors of linear predictors are computed after subtracting the original column means from the new design matrix.

conf.int — Specify conf.int as a positive fraction to obtain upper and lower confidence intervals (e.g., conf.int=0.95). The t-distribution is used in the calculation for ols fits. Otherwise, the normal critical value is used. For Bayesian models conf.int is the highest posterior density interval probability.

conf.type — specifies the type of confidence interval. Default is for the mean. For ols fits there is the option of obtaining confidence limits for individual predicted values by specifying conf.type="individual".

posterior.summary — when making predictions from a Bayesian model, specifies whether you want the linear predictor to be computed from the posterior mean of parameters (default) or the posterior mode or median median

second — set to TRUE to use the model's second formula. At present this pertains only to a partial proportional odds model fitted using the blrm function. When second=TRUE and type='x' the Z design matrix is returned (that goes with the tau parameters in the partial PO model). When type='lp' is specified Z*tau is computed. In neither case is the result is multiplied by the by the cppo function.

kint — a single integer specifying the number of the intercept to use in multiple-intercept models. The default is 1 for lrm and the reference median intercept for orm and blrm. For a partial PO model, kint should correspond to the response variable value that will be used when dealing with second=TRUE.

na.action — Function to handle missing values in newdata. For predictions "in data", the same na.action that was used during model fitting is used to define an naresid function to possibly restore rows of the data matrix that were deleted due to NAs.

For predictions "out of data", the default na.action is na.keep, resulting in NA predictions when a row of newdata has an NA. Whatever na.action is in effect at the time for "out of data" predictions, the corresponding naresid is used also.

expand.na — set to FALSE to keep the naresid from having any effect, i.e., to keep from adding back observations removed because of NAs in the returned object. If expand.na=FALSE, the na.action attribute will be added to the returned object.

center.terms — set to FALSE to suppress subtracting adjust-to values from columns of the design matrix before computing terms with type="terms".

ref.zero — Set to TRUE to subtract a constant from $X\beta$ before plotting so that the reference value of the x-variable yields y=0. This is done before applying function fun. This is especially useful for Cox models to make the hazard ratio be 1.0 at reference values, and the confidence interval have width zero.

... — ignored

Details

datadist and options(datadist=) should be run before predictrms if using type="adjto", type="adjto.data.frame", or type="terms", or if the fit is a Cox model fit and you are requesting se.fit=TRUE. For these cases, the adjustment values are needed (either for the returned result or for the correct covariance matrix computation).

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

plot.Predict, ggplot.Predict, summary.rms, rms, rms.trans, predict.lrm, predict.orm, residuals.cph, datadist, gendata, gIndex, Function.rms, reShape, xYplot, contrast.rms

Examples

```
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
treat          <- factor(sample(c('a','b','c'), n,TRUE))


# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
  (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male')) +
  .3*sqrt(blood.pressure-60)-2.3 + 1*(treat=='b')
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)
```

```
ddist <- datadist(age, blood.pressure, cholesterol, sex, treat)
options(datadist='ddist')


fit <- lrm(y ~ rcs(blood.pressure,4) +
           sex * (age + rcs(cholesterol,4)) + sex*treat*age)


# Use xYplot to display predictions in 9 panels, with error bars,
# with superposition of two treatments


dat <- expand.grid(treat=levels(treat),sex=levels(sex),
                   age=c(20,40,60),blood.pressure=120,
                   cholesterol=seq(100,300,length=10))
# Add variables linear.predictors and se.fit to dat
dat <- cbind(dat, predict(fit, dat, se.fit=TRUE))
# This is much easier with Predict
# xYplot in Hmisc extends xyplot to allow error bars

xYplot(Cbind(linear.predictors,linear.predictors-1.96*se.fit,
             linear.predictors+1.96*se.fit) ~ cholesterol | sex*age,
       groups=treat, data=dat, type='b')




# Since blood.pressure doesn't interact with anything, we can quickly and
# interactively try various transformations of blood.pressure, taking
# the fitted spline function as the gold standard. We are seeking a
# linearizing transformation even though this may lead to falsely
# narrow confidence intervals if we use this data-dredging-based transformation


bp <- 70:160
logit <- predict(fit, expand.grid(treat="a", sex='male', age=median(age),
                 cholesterol=median(cholesterol),
                 blood.pressure=bp), type="terms")[,"blood.pressure"]
#Note: if age interacted with anything, this would be the age
#      "main effect" ignoring interaction terms
#Could also use Predict(f, age=ag)$yhat
#which allows evaluation of the shape for any level of interacting
#factors.  When age does not interact with anything, the result from
#predict(f, \dots, type="terms") would equal the result from
#plot if all other terms were ignored


plot(bp^.5, logit)               # try square root vs. spline transform.
plot(bp^1.5, logit)              # try 1.5 power
plot(sqrt(bp-60), logit)
```

```
#Some approaches to making a plot showing how predicted values
#vary with a continuous predictor on the x-axis, with two other
#predictors varying


combos <- gendata(fit, age=seq(10,100,by=10), cholesterol=c(170,200,230),
                  blood.pressure=c(80,120,160))
#treat, sex not specified -> set to mode
#can also used expand.grid

require(lattice)
combos$pred <- predict(fit, combos)
xyplot(pred ~ age | cholesterol*blood.pressure, data=combos, type='l')
xYplot(pred ~ age | cholesterol, groups=blood.pressure, data=combos, type='l')
Key()   # Key created by xYplot
xYplot(pred ~ age, groups=interaction(cholesterol,blood.pressure),
       data=combos, type='l', lty=1:9)
Key()


# Add upper and lower 0.95 confidence limits for individuals
combos <- cbind(combos, predict(fit, combos, conf.int=.95))
xYplot(Cbind(linear.predictors, lower, upper) ~ age | cholesterol,
       groups=blood.pressure, data=combos, type='b')
Key()


# Plot effects of treatments (all pairwise comparisons) vs.
# levels of interacting factors (age, sex)


d <- gendata(fit, treat=levels(treat), sex=levels(sex), age=seq(30,80,by=10))
x <- predict(fit, d, type="x")
betas <- fit$coef
cov   <- vcov(fit, intercepts='none')


i <- d$treat=="a"; xa <- x[i,]; Sex <- d$sex[i]; Age <- d$age[i]
i <- d$treat=="b"; xb <- x[i,]
i <- d$treat=="c"; xc <- x[i,]


doit <- function(xd, lab) {
  xb <- matxv(xd, betas)
  se <- apply((xd %*% cov) * xd, 1, sum)^.5
  q <- qnorm(1-.01/2)   # 0.99 confidence limits
  lower <- xb - q * se; upper <- xb + q * se
  #Get odds ratios instead of linear effects
  xb <- exp(xb); lower <- exp(lower); upper <- exp(upper)
  #First elements of these agree with
  #summary(fit, age=30, sex='female',conf.int=.99))
  for(sx in levels(Sex)) {
```

```
    j <- Sex==sx
    errbar(Age[j], xb[j], upper[j], lower[j], xlab="Age",
           ylab=paste(lab, "Odds Ratio"), ylim=c(.1, 20), log='y')
    title(paste("Sex:", sx))
    abline(h=1, lty=2)
  }
}


par(mfrow=c(3,2), oma=c(3,0,3,0))
doit(xb - xa, "b:a")
doit(xc - xa, "c:a")
doit(xb - xa, "c:b")

# NOTE: This is much easier to do using contrast.rms

# Demonstrate type="terms", "cterms", "ccterms"
set.seed(1)
n <- 40
x <- 1:n
w <- factor(sample(c('a', 'b'), n, TRUE))
u <- factor(sample(c('A', 'B'), n, TRUE))
y <- .01*x + .2*(w=='b') + .3*(u=='B') + .2*(w=='b' & u=='B') + rnorm(n)/5
ddist <- datadist(x, w, u)
f <- ols(y ~ x*w*u, x=TRUE, y=TRUE)
f
anova(f)
z <- predict(f, type='terms', center.terms=FALSE)
z[1:5,]
k <- coef(f)
## Manually compute combined terms
wb <- w=='b'
uB <- u=='B'
h  <- k['x * w=b * u=B']*x*wb*uB
tx <- k['x']  *x  + k['x * w=b']*x*wb + k['x * u=B']  *x*uB  + h
tw <- k['w=b']*wb + k['x * w=b']*x*wb + k['w=b * u=B']*wb*uB + h
tu <- k['u=B']*uB + k['x * u=B']*x*uB + k['w=b * u=B']*wb*uB + h
h   <- z[,'x * w * u'] # highest order term is present in all cterms
tx2 <- z[,'x']+z[,'x * w']+z[,'x * u']+h
tw2 <- z[,'w']+z[,'x * w']+z[,'w * u']+h
tu2 <- z[,'u']+z[,'x * u']+z[,'w * u']+h
ae <- function(a, b) all.equal(a, b, check.attributes=FALSE)
ae(tx, tx2)
ae(tw, tw2)
ae(tu, tu2)

zc <- predict(f, type='cterms')
zc[1:5,]
ae(tx, zc[,'x'])
ae(tw, zc[,'w'])
ae(tu, zc[,'u'])

zc <- predict(f, type='ccterms')
```

```
# As all factors are indirectly related, ccterms gives overall linear
# predictor except for the intercept
zc[1:5,]
ae(as.vector(zc + coef(f)[1]), f$linear.predictors)

## Not run:
#A variable state.code has levels "1", "5","13"
#Get predictions with or without converting variable in newdata to factor
predict(fit, data.frame(state.code=c(5,13)))
predict(fit, data.frame(state.code=factor(c(5,13))))


#Use gendata function (gendata.rms) for interactive specification of
#predictor variable settings (for 10 observations)
df <- gendata(fit, nobs=10, viewvals=TRUE)
df$predicted <- predict(fit, df)  # add variable to data frame
df


df <- gendata(fit, age=c(10,20,30))  # leave other variables at ref. vals.
predict(fit, df, type="fitted")


# See reShape (in Hmisc) for an example where predictions corresponding to
# values of one of the varying predictors are reformatted into multiple
# columns of a matrix

## End(Not run)
options(datadist=NULL)
```

`print.cph` *Print cph Results*

Description

Formatted printing of an object of class cph. Prints strata frequencies, parameter estimates, standard errors, z-statistics, numbers of missing values, etc. Format of output is controlled by the user previously running `options(prType="lang")` where `lang` is `"plain"` (the default), `"latex"`, or `"html"`. This does not require `results='asis'` in `knitr` chunk headers.

Usage

```
## S3 method for class 'cph'
print(x, digits=4, r2=c(0,2,4), table=TRUE, conf.int=FALSE,
coefs=TRUE, pg=FALSE, title='Cox Proportional Hazards Model', ...)
```

Arguments

x	fit object

digits	number of digits to right of decimal place to print
r2	vector of integers specifying which R^2 measures to print, with 0 for Nagelkerke R^2 and 1:4 corresponding to the 4 measures computed by R2Measures. Default is to print Nagelkerke (labeled R2) and second and fourth R2Measures which are the measures adjusted for the number of predictors, first for the raw sample size then for the effective sample size, which here is the number of non-censored observations.
conf.int	set to e.g. .95 to print 0.95 confidence intervals on simple hazard ratios (which are usually meaningless as one-unit changes are seldom relevant and most models contain multiple terms per predictor)
table	set to FALSE to suppress event frequency statistics
coefs	specify coefs=FALSE to suppress printing the table of model coefficients, standard errors, etc. Specify coefs=n to print only the first n regression coefficients in the model.
pg	set to TRUE to print g-indexes
title	a character string title to be passed to prModFit
...	arguments passed to prModFit

See Also

coxph, prModFit

---

print.Glm *print.glm*

Description

Print a ‘Glm‘ Object

Usage

```
## S3 method for class 'Glm'
print(x, digits = 4, coefs = TRUE, title = "General Linear Model", ...)
```

Arguments

x	‘Glm‘ object
digits	number of significant digits to print
coefs	specify ‘coefs=FALSE‘ to suppress printing the table of model coefficients, standard errors, etc. Specify ‘coefs=n‘ to print only the first ‘n‘ regression coefficients in the model.
title	a character string title to be passed to ‘prModFit‘
...	ignored

Details

Prints a ‘Glm‘ object, optionally in LaTeX or html

Author(s)

Frank Harrell

---

print.impactPO *Print Result from impactPO*

Description

Prints statistical summaries and optionally predicted values computed by `impactPO`, transposing statistical summaries for easy reading

Usage

```
## S3 method for class 'impactPO'
print(x, estimates = nrow(x$estimates) < 16, ...)
```

Arguments

`x`	an object created by `impactPO`
`estimates`	set to `FALSE` to suppess printing estimated category probabilities. Defaults to `TRUE` when the number of rows < 16.
`...`	ignored

Author(s)

Frank Harrell

---

print.Ocens *print Method for Ocens Objects*

Description

print Method for Ocens Objects

Usage

```
## S3 method for class 'Ocens'
print(x, ivalues = FALSE, digits = 5, ...)
```

Arguments

x	an object created by Ocens
ivalues	set to TRUE to print integer codes instead of character levels when original data were factors or character variables
digits	number of digits to the right of the decimal place used in rounding original levels when ivalues=FALSE
...	ignored

Value

nothing

Examples

```
Y <- Ocens(1:3, c(1, Inf, 3))
Y
print(Y, ivalues=TRUE)  # doesn't change anything since were numeric
```

---

print.ols *Print ols*

Description

Formatted printing of an object of class ols using methods taken from print.lm and summary.lm. Prints R-squared, adjusted R-squared, parameter estimates, standard errors, and t-statistics (Z statistics if penalized estimation was used). For penalized estimation, prints the maximum penalized likelihood estimate of the residual standard deviation (Sigma) instead of the usual root mean squared error. Format of output is controlled by the user previously running options(prType="lang") where lang is "plain" (the default), "latex", or "html". When using html with Quarto or RMarkdown, results='asis' need not be written in the chunk header.

Usage

```
## S3 method for class 'ols'
print(x, digits=4, long=FALSE, coefs=TRUE,
 title="Linear Regression Model", ...)
```

Arguments

x	fit object
digits	number of significant digits to print
long	set to TRUE to print the correlation matrix of parameter estimates
coefs	specify coefs=FALSE to suppress printing the table of model coefficients, standard errors, etc. Specify coefs=n to print only the first n regression coefficients in the model.
title	a character string title to be passed to prModFit
...	other parameters to pass to print or format

See Also

ols, lm,prModFit

---

print.rexVar *print.rexVar*

Description

Print rexVar Result

Usage

```
## S3 method for class 'rexVar'
print(x, title = "Relative Explained Variation", digits = 3, ...)
```

Arguments

x	a vector or matrix created by rexVar
title	character string which can be set to NULL or '' to suppress
digits	passed to round()
...	unused

Details

Prints the results of an rexVar call

Value

invisible

Author(s)

Frank Harrell

`prmiInfo` *prmiInfo*

Description

Print Information About Impact of Imputation

Usage

```
prmiInfo(x)
```

Arguments

`x`	an object created by `processMI(..., 'anova')`

Details

For the results of `processMI.fit.mult.impute` prints or writes html (the latter if `options(prType='html')` is in effect) summarizing various correction factors related to missing data multiple imputation.

Value

nothing

Author(s)

Frank Harrell

Examples

```
## Not run:
a <- aregImpute(...)
f <- fit.mult.impute(...)
v <- processMI(f, 'anova')
prmiInfo(v)

## End(Not run)
```

processMI *processMI*

Description

Process Special Multiple Imputation Output

Usage

```
processMI(object, ...)
```

Arguments

object	a fit object created by Hmisc::fit.mult.impute()
...	ignored

Details

Processes lists that have one element per imputation

Value

an object that resembles something created by a single fit without multiple imputation

Author(s)

Frank Harrell

See Also

processMI.fit.mult.impute()

processMI.fit.mult.impute *processMI.fit.mult.impute*

Description

Process Special Multiple Imputation Output From fit.mult.impute

Usage

```
## S3 method for class 'fit.mult.impute'
processMI(
  object,
  which = c("validate", "calibrate", "anova"),
  plotall = TRUE,
  nind = 0,
  prmi = TRUE,
  ...
)
```

Arguments

object	a fit object created by fit.mult.impute
which	specifies which component of the extra output should be processed
plotall	set to FALSE when which='calibrate' to suppress having ggplot render a graph showing calibration curves produced separately for all the imputations
nind	set to a positive integer to use base graphics to plot a matrix of graphs, one each for the first nind imputations, and the overall average calibration curve at the end
prmi	set to FALSE to not print imputation corrections for anova
...	ignored

Details

Processes a funresults object stored in a fit object created by fit.mult.impute when its fun argument was used. These objects are typically named validate or calibrate and represent bootstrap or cross-validations run separately for each imputation. See this for a case study.

For which='anova' assumes that the fun given to fit.mult.impute runs anova(fit, test='LR') to get likelihood ratio tests, and that method='stack' was specified also so that a final anova was run on the stacked combination of all completed datasets. The method of Chan and Meng (2022) is used to obtain overall likelihood ratio tests, with each line of the anova table getting a customized adjustment based on the amount of missing information pertaining to the variables tested in that line. The resulting statistics are chi-square and not F statistics as used by Chan and Meng. This will matter when the estimated denominator degrees of freedom for a variable is small (e.g., less than 50). These d.f. are reported so that user can take appropriate cautions such as increasing n.impute for aregImpute.

Value

an object like a validate, calibrate, or anova result obtained when no multiple imputation was done. This object is suitable for print and plot methods for these kinds of objects.

Author(s)

Frank Harrell

See Also

Hmisc::fit.mult.impute()

---

psm *Parametric Survival Model*

Description

psm is a modification of Therneau's survreg function for fitting the accelerated failure time family of parametric survival models. psm uses the rms class for automatic anova, fastbw, calibrate, validate, and other functions. Hazard.psm, Survival.psm, Quantile.psm, and Mean.psm create S functions that evaluate the hazard, survival, quantile, and mean (expected value) functions analytically, as functions of time or probabilities and the linear predictor values. The Nagelkerke R^2 and and adjusted Maddala-Cox-Snell R^2 are computed. For the latter the notation is R2(p,m) where p is the number of regression coefficients being adjusted for and m is the effective sample size (number of uncensored observations). See R2Measures for more information.

For the print method, format of output is controlled by the user previously running options(prType="lang") where lang is "plain" (the default), "latex", or "html".

The residuals.psm function exists mainly to compute normalized (standardized) residuals and to censor them (i.e., return them as Surv objects) just as the original failure time variable was censored. These residuals are useful for checking the underlying distributional assumption (see the examples). To get these residuals, the fit must have specified y=TRUE. A lines method for these residuals automatically draws a curve with the assumed standardized survival distribution. A survplot method runs the standardized censored residuals through npsurv to get Kaplan-Meier estimates, with optional stratification (automatically grouping a continuous variable into quantiles) and then through survplot.npsurv to plot them. Then lines is invoked to show the theoretical curve. Other types of residuals are computed by residuals using residuals.survreg.

Usage

```
psm(formula,
    data=environment(formula), weights,
    subset, na.action=na.delete, dist="weibull",
    init=NULL, scale=0,
    control=survreg.control(),
    parms=NULL,
    model=FALSE, x=FALSE, y=TRUE, time.inc, ...)

## S3 method for class 'psm'
print(x, correlation=FALSE, digits=4, r2=c(0,2,4), coefs=TRUE,
pg=FALSE, title, ...)

Hazard(object, ...)
## S3 method for class 'psm'
Hazard(object, ...)   # for psm fit
# E.g. lambda <- Hazard(fit)
```

```
Survival(object, ...)
## S3 method for class 'psm'
Survival(object, ...) # for psm
# E.g. survival <- Survival(fit)

## S3 method for class 'psm'
Quantile(object, ...) # for psm
# E.g. quantsurv <- Quantile(fit)

## S3 method for class 'psm'
Mean(object, ...)     # for psm
# E.g. meant   <- Mean(fit)

# lambda(times, lp)   # get hazard function at t=times, xbeta=lp
# survival(times, lp) # survival function at t=times, lp
# quantsurv(q, lp)    # quantiles of survival time
# meant(lp)           # mean survival time

## S3 method for class 'psm'
residuals(object, type=c("censored.normalized",
"response", "deviance", "dfbeta",
"dfbetas", "working", "ldcase", "ldresp", "ldshape", "matrix", "score"), ...)

## S3 method for class 'residuals.psm.censored.normalized'
survplot(fit, x, g=4, col, main, ...)

## S3 method for class 'residuals.psm.censored.normalized'
lines(x, n=100, lty=1, xlim,
lwd=3, ...)
# for type="censored.normalized"
```

Arguments

formula	an S statistical model formula. Interactions up to third order are supported. The left hand side must be a Surv object.
object	a fit created by psm. For survplot with residuals from psm, object is the result of residuals.psm.
fit	a fit created by psm
data, subset, weights, dist, scale, init, na.action, control	see survreg.
parms	a list of fixed parameters. For the t-distribution this is the degrees of freedom; most of the distributions have no parameters.
model	set to TRUE to include the model frame in the returned object
x	set to TRUE to include the design matrix in the object produced by psm. For the survplot method, x is an optional stratification variable (character, numeric, or categorical). For lines.residuals.psm.censored.normalized, x is the result of residuals.psm. For print it is the result of psm.

y	set to TRUE to include the Surv() matrix
time.inc	setting for default time spacing. Used in constructing time axis in survplot, and also in make confidence bars. Default is 30 if time variable has units="Day", 1 otherwise, unless maximum follow-up time < 1. Then max time/10 is used as time.inc. If time.inc is not given and max time/default time.inc is > 25, time.inc is increased.
correlation	set to TRUE to print the correlation matrix for parameter estimates
digits	number of places to print to the right of the decimal point
r2	vector of integers specifying which R^2 measures to print, with 0 for Nagelkerke R^2 and 1:4 corresponding to the 4 measures computed by R2Measures. Default is to print Nagelkerke (labeled R2) and second and fourth R2Measures which are the measures adjusted for the number of predictors, first for the raw sample size then for the effective sample size, which here is the number of uncensored observations.
coefs	specify coefs=FALSE to suppress printing the table of model coefficients, standard errors, etc. Specify coefs=n to print only the first n regression coefficients in the model.
pg	set to TRUE to print g-indexes
title	a character string title to be passed to prModFit
...	other arguments to fitting routines, or to pass to survplot from survplot.residuals.psm.censored.normalized. Passed to the generic lines function for lines.
times	a scalar or vector of times for which to evaluate survival probability or hazard
lp	a scalar or vector of linear predictor values at which to evaluate survival probability or hazard. If both times and lp are vectors, they must be of the same length.
q	a scalar or vector of probabilities. The default is .5, so just the median survival time is returned. If q and lp are both vectors, a matrix of quantiles is returned, with rows corresponding to lp and columns to q.
type	type of residual desired. Default is censored normalized residuals, defined as (link(Y) - linear.predictors)/scale parameter, where the link function was usually the log function. See survreg for other types. type="score" returns the score residual matrix.
n	number of points to evaluate theoretical standardized survival function for lines.residuals.psm.censored.normalized
lty	line type for lines, default is 1
xlim	range of times (or transformed times) for which to evaluate the standardized survival function. Default is range in normalized residuals.
lwd	line width for theoretical distribution, default is 3
g	number of quantile groups to use for stratifying continuous variables having more than 5 levels
col	vector of colors for survplot method, corresponding to levels of x (must be a scalar if there is no x)
main	main plot title for survplot. If omitted, is the name or label of x if x is given. Use main="" to suppress a title when you specify x.

Details

The object survreg.distributions contains definitions of properties of the various survival distributions.
psm does not trap singularity errors due to the way survreg.fit does matrix inversion. It will trap non-convergence (thus returning fit$fail=TRUE) if you give the argument failure=2 inside the control list which is passed to survreg.fit. For example, use f <- psm(S ~ x, control=list(failure=2, maxiter=20)) to allow up to 20 iterations and to set f$fail=TRUE in case of non-convergence. This is especially useful in simulation work.

Value

psm returns a fit object with all the information survreg would store as well as what rms stores and units and time.inc. Hazard, Survival, and Quantile return S-functions. residuals.psm with type="censored.normalized" returns a Surv object which has a special attribute "theoretical" which is used by the lines routine. This is the assumed standardized survival function as a function of time or transformed time.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
<fh@fharrell.com>

See Also

rms, survreg, residuals.survreg, survreg.object, survreg.distributions, pphsm, survplot, survest, Surv, na.delete, na.detail.response, datadist, latex.psm, GiniMd, prModFit, ggplot.Predict, plot.Predict, R2Measures

Examples

```
require(survival)
n <- 400
set.seed(1)
age <- rnorm(n, 50, 12)
sex <- factor(sample(c('Female','Male'),n,TRUE))
dd <- datadist(age,sex)
options(datadist='dd')
# Population hazard function:
h <- .02*exp(.06*(age-50)+.8*(sex=='Female'))
d.time <- -log(runif(n))/h
cens <- 15*runif(n)
death <- ifelse(d.time <= cens,1,0)
d.time <- pmin(d.time, cens)

f <- psm(Surv(d.time,death) ~ sex*pol(age,2),
         dist='lognormal')
# Log-normal model is a bad fit for proportional hazards data
print(f, r2=0:4, pg=TRUE)
```

```
anova(f)
fastbw(f)  # if deletes sex while keeping age*sex ignore the result
f <- update(f, x=TRUE,y=TRUE)       # so can validate, compute certain resids
validate(f, B=10)      # ordinarily use B=300 or more
plot(Predict(f, age, sex))   # needs datadist since no explicit age, hosp.
# Could have used ggplot(Predict(...))
survplot(f, age=c(20,60))     # needs datadist since hospital not set here
# latex(f)


S <- Survival(f)
plot(f$linear.predictors, S(6, f$linear.predictors),
     xlab=expression(X*hat(beta)),
     ylab=expression(S(6,X*hat(beta))))
# plots 6-month survival as a function of linear predictor (X*Beta hat)


times <- seq(0,24,by=.25)
plot(times, S(times,0), type='l')   # plots survival curve at X*Beta hat=0
lam <- Hazard(f)
plot(times, lam(times,0), type='l') # similarly for hazard function


med <- Quantile(f)        # new function defaults to computing median only
lp <- seq(-3, 5, by=.1)
plot(lp, med(lp=lp), ylab="Median Survival Time")
med(c(.25,.5), f$linear.predictors)
                          # prints matrix with 2 columns


# fit a model with no predictors
f <- psm(Surv(d.time,death) ~ 1, dist="weibull")
f
pphsm(f)          # print proportional hazards form
g <- survest(f)
plot(g$time, g$surv, xlab='Time', type='l',
     ylab=expression(S(t)))


f <- psm(Surv(d.time,death) ~ age,
         dist="loglogistic", y=TRUE)
r <- resid(f, 'cens') # note abbreviation
survplot(npsurv(r ~ 1), conf='none')
                      # plot Kaplan-Meier estimate of
                      # survival function of standardized residuals
survplot(npsurv(r ~ cut2(age, g=2)), conf='none')
                      # both strata should be n(0,1)
lines(r)              # add theoretical survival function
#More simply:
survplot(r, age, g=2)

options(datadist=NULL)
```

Punits *Prepare units for Printing and Plotting*

Description

Takes a character variable containing units of measurement for a variable. If it has zero length, a "" string is return. Otherwise, any trailing "s" is removed if the string is longer than one character, and depending on the arguments, the string is changed to lower case, "s" is added, and the first character is changed to upper case.

Usage

```
Punits(u, lower = TRUE, adds = TRUE, upfirst = FALSE, default = "")
```

Arguments

u	a single string containing units of measurement
lower	if TRUE set string to all lower case
adds	if TRUE add trailing "s"
upfirst	if TRUE set first character to upper case
default	default units if u is empty

Value

a single character string

See Also

Hmisc::units()

Examples

```
## Not run:
Punits('Years')

## End(Not run)
```

recode2integer *recode2integer*

Description

Create Ordinal Variables With a Given Precision

Usage

```
recode2integer(y, precision = 7, ftable = TRUE)
```

Arguments

y	a numeric, factor, or character vector with no NAs
precision	number of places to the right of the decimal place to round y if y is numeric but not integer, for the purpose of finding the distinct values. Real values rounding to the same values under precision are mapped to the same integer output y
ftable	set to FALSE to suppress creation of freq

Details

For a factor variable y, uses existing factor levels and codes the output y as integer. For a character y, converts to factor and does the same. For a numeric y that is integer, leaves the levels intact and codes y as consecutive positive integers corresponding to distinct values in the data. For numeric y that contains any non-integer values, rounds y to precision decimal places to the right before finding the distinct values.

This function is used to prepare ordinal variables for orm.fit() and lrm.fit(). It was written because just using factor() creates slightly different distinct y levels on different hardware because factor() uses unique() which functions slightly differently on different systems when there are non-significant digits in floating point numbers. See this for more details.

Value

a list with the following elements:

- y: vector of integer-coded y
- ylevels: vector of corresponding original y values, possibly rounded to precision. This vector is numeric unless y is factor or character, in which case it is a character vector.
- freq: frequency table of rounded or categorical y, with names attribute for the (possibly rounded) y levels of the frequencies
- median: median y from original values if numeric, otherwise median of the new integer codes for y
- whichmedian: the integer valued y that most closely corresponds to median; for an ordinal regression model this represents one plus the index of the intercept vector corresponding to median.

Author(s)

Cole Beck

Examples

```
w <- function(y, precision=7) {
  v <- recode2integer(y, precision);
  print(v)
  print(table(y, ynew=v$y))
}
set.seed(1)
w(sample(1:3, 20, TRUE))
w(sample(letters[1:3], 20, TRUE))
y <- runif(20)
w(y)
w(y, precision=2)
```

---

residuals.cph *Residuals for a cph Fit*

Description

Calculates martingale, deviance, score or Schoenfeld residuals (scaled or unscaled) or influence statistics for a Cox proportional hazards model. This is a slightly modified version of Therneau's `residuals.coxph` function. It assumes that `x=TRUE` and `y=TRUE` were specified to `cph`, except for martingale residuals, which are stored with the fit by default.

Usage

```
## S3 method for class 'cph'
residuals(object,
      type=c("martingale", "deviance", "score", "schoenfeld",
             "dfbeta", "dfbetas", "scaledsch", "partial"), ...)
```

Arguments

`object`	a cph object
`type`	character string indicating the type of residual desired; the default is martingale. Only enough of the string to determine a unique match is required. Instead of the usual residuals, `type="dfbeta"` may be specified to obtain approximate leave-out-one $\Delta\beta$s. Use `type="dfbetas"` to normalize the $\Delta\beta$s for the standard errors of the regression coefficient estimates. Scaled Schoenfeld residuals (`type="scaledsch"`, Grambsch and Therneau, 1993) better reflect the log hazard ratio function than ordinary Schoenfeld residuals, and they are on the regression coefficient scale. The weights use Grambsch and Therneau's "average variance" method.
`...`	see `residuals.coxph`

Value

The object returned will be a vector for martingale and deviance residuals and matrices for score and schoenfeld residuals, dfbeta, or dfbetas. There will be one row of residuals for each row in the input data (without `collapse`). One column of score and Schoenfeld residuals will be returned for each column in the model.matrix. The scaled Schoenfeld residuals are used in the `cox.zph` function.

The score residuals are each individual's contribution to the score vector. Two transformations of this are often more useful: `dfbeta` is the approximate change in the coefficient vector if that observation were dropped, and `dfbetas` is the approximate change in the coefficients, scaled by the standard error for the coefficients.

References

T. Therneau, P. Grambsch, and T.Fleming. "Martingale based residuals for survival models", Biometrika, March 1990.

P. Grambsch, T. Therneau. "Proportional hazards tests and diagnostics based on weighted residuals", unpublished manuscript, Feb 1993.

See Also

`cph`, `coxph`, `residuals.coxph`, `cox.zph`, `naresid`

Examples

```
# fit <- cph(Surv(start, stop, event) ~ (age + surgery)* transplant,
#            data=jasa1)
# mresid <- resid(fit, collapse=jasa1$id)


# Get unadjusted relationships for several variables
# Pick one variable that's not missing too much, for fit

require(survival)
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
cens   <- 15*runif(n)
h      <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
d.time <- -log(runif(n))/h
death  <- ifelse(d.time <= cens,1,0)
d.time <- pmin(d.time, cens)


f <- cph(Surv(d.time, death) ~ age + blood.pressure + cholesterol, iter.max=0)
res <- resid(f) # This re-inserts rows for NAs, unlike f$resid
yl <- quantile(res, c(10/length(res),1-10/length(res)), na.rm=TRUE)
# Scale all plots from 10th smallest to 10th largest residual
```

```
par(mfrow=c(2,2), oma=c(3,0,3,0))
p <- function(x) {
  s <- !is.na(x+res)
  plot(lowess(x[s], res[s], iter=0), xlab=label(x), ylab="Residual",
       ylim=yl, type="l")
}
p(age); p(blood.pressure); p(cholesterol)
mtext("Smoothed Martingale Residuals", outer=TRUE)


# Assess PH by estimating log relative hazard over time
f <- cph(Surv(d.time,death) ~ age + sex + blood.pressure, x=TRUE, y=TRUE)
r <- resid(f, "scaledsch")
tt <- as.numeric(dimnames(r)[[1]])
par(mfrow=c(3,2))
for(i in 1:3) {
  g <- areg.boot(I(r[,i]) ~ tt, B=20)
  plot(g, boot=FALSE)  # shows bootstrap CIs
}                  # Focus on 3 graphs on right
# Easier approach:
plot(cox.zph(f))    # invokes plot.cox.zph
par(mfrow=c(1,1))
```

`residuals.Glm` *residuals.Glm*

Description

Residuals for ‘Glm‘

Usage

```
## S3 method for class 'Glm'
residuals(object, type, ...)
```

Arguments

`object`	a fit object produced by ‘Glm‘
`type`	either ‘’score’‘ or a ‘type‘ accepted by ‘residuals.glm‘
`...`	ignored

Details

This function mainly passes through to ‘residuals.glm‘ but for ‘type=’score’‘ computes the matrix of score residuals using code modified from ‘sandwich::estfun.glm‘.

Value

a vector or matrix

Author(s)

Frank Harrell

`residuals.lrm` *Residuals from an* `lrm` *or* `orm` *Fit*

Description

For a binary logistic model fit, computes the following residuals, letting P denote the predicted probability of the higher category of Y, X denote the design matrix (with a column of 1s for the intercept), and L denote the logit or linear predictors: ordinary or Li-Shepherd ($Y - P$), score ($X(Y-P)$), pearson ($(Y-P)/\sqrt{P(1-P)}$), deviance (for $Y = 0$ is $-\sqrt{2|\log(1-P)|}$, for $Y = 1$ is $\sqrt{2|\log(P)|}$, pseudo dependent variable used in influence statistics ($L + (Y - P)/(P(1 - P))$), and partial ($X_i\beta_i + (Y - P)/(P(1 - P))$).

Will compute all these residuals for an ordinal logistic model, using as temporary binary responses dichotomizations of Y, along with the corresponding P, the probability that $Y \geq$ cutoff. For `type="partial"`, all possible dichotomizations are used, and for `type="score"`, the actual components of the first derivative of the log likelihood are used for an ordinal model. For `type="li.shepherd"` the residual is $Pr(W < Y) - Pr(W > Y)$ where Y is the observed response and W is a random variable from the fitted distribution. Alternatively, specify `type="score.binary"` to use binary model score residuals but for all cutpoints of Y (plotted only, not returned). The `score.binary`, `partial`, and perhaps `score` residuals are useful for checking the proportional odds assumption although many attempts to do so have failed. If the option `pl=TRUE` is used to plot the `score` or `score.binary` residuals, a score residual plot is made for each column of the design (predictor) matrix, with `Y` cutoffs on the x-axis and the mean +- 1.96 standard errors of the score residuals on the y-axis. You can instead use a box plot to display these residuals, for both `score.binary` and `score`. Proportional odds dictates a horizontal `score.binary` plot. Partial residual plots use smooth nonparametric estimates, separately for each cutoff of Y. One examines that plot for parallelism of the curves to check the proportional odds assumption, as well as to see if the predictor behaves linearly.

Also computes a variety of influence statistics and the le Cessie - van Houwelingen - Copas - Hosmer unweighted sum of squares test for global goodness of fit, done separately for each cutoff of Y in the case of an ordinal model.

The `plot.lrm.partial` function computes partial residuals for a series of binary logistic model fits that all used the same predictors and that specified `x=TRUE, y=TRUE`. It then computes smoothed partial residual relationships (using `lowess` with `iter=0`) and plots them separately for each predictor, with residual plots from all model fits shown on the same plot for that predictor.

Usage

```
## S3 method for class 'lrm'
residuals(object, type=c("li.shepherd","ordinary",
 "score", "score.binary", "pearson", "deviance", "pseudo.dep",
 "partial", "dfbeta", "dfbetas", "dffit", "dffits", "hat", "gof", "lp1"),
           pl=FALSE, xlim, ylim, kint, label.curves=TRUE, which, ...)
```

```
## S3 method for class 'orm'
residuals(object, type=c("li.shepherd","ordinary",
 "score", "score.binary", "pearson", "deviance", "pseudo.dep",
 "partial", "dfbeta", "dfbetas", "dffit", "dffits", "hat", "gof", "lp1"),
           pl=FALSE, xlim, ylim, kint, label.curves=TRUE, which, ...)

## S3 method for class 'lrm.partial'
plot(..., labels, center=FALSE, ylim)
```

Arguments

object	object created by lrm or orm
...	for residuals, applies to type="partial" when pl is not FALSE. These are extra arguments passed to the smoothing function. Can also be used to pass extra arguments to boxplot for type="score" or "score.binary". For plot.lrm.partial this specifies a series of binary model fit objects.
type	type of residual desired. Use type="lp1" to get approximate leave-out-1 linear predictors, derived by subtracting the dffit from the original linear predictor values.
pl	applies only to type="partial", "score", and "score.binary". For score residuals in an ordinal model, set pl=TRUE to get means and approximate 0.95 confidence bars vs. Y, separately for each X. Alternatively, specify pl="boxplot" to use boxplot to draw the plot, with notches and with width proportional to the square root of the cell sizes. For partial residuals, set pl=TRUE (which uses lowess) or pl="supsmu" to get smoothed partial residual plots for all columns of X using supsmu. Use pl="loess" to use loess and get confidence bands ("loess" is not implemented for ordinal responses). Under R, pl="loess" uses lowess and does not provide confidence bands. If there is more than one X, you should probably use par(mfrow=c(,)) before calling resid. Note that pl="loess" results in plot.loess being called, which requires a large memory allocation.
xlim	plotting range for x-axis (default = whole range of predictor)
ylim	plotting range for y-axis (default = whole range of residuals, range of all confidence intervals for score or score.binary or range of all smoothed curves for partial if pl=TRUE, or 0.1 and 0.9 quantiles of the residuals for pl="boxplot".)
kint	for an ordinal model for residuals other than li.shepherd, partial, score, or score.binary, specifies the intercept (and the cutoff of Y) to use for the calculations. Specifying kint=2, for example, means to use $Y \geq$ 3rd level.
label.curves	set to FALSE to suppress curve labels when type="partial". The default, TRUE, causes labcurve to be invoked to label curves where they are most separated. label.curves can be a list containing the opts parameter for labcurve, to send options to labcurve, such as tilt. The default for tilt here is TRUE.
which	a vector of integers specifying column numbers of the design matrix for which to compute or plot residuals, for type="partial","score","score.binary".
labels	for plot.lrm.partial this specifies a vector of character strings providing labels for the list of binary fits. By default, the names of the fit objects are used as labels. The labcurve function is used to label the curve with the labels.

`center` for `plot.lrm.partial` this causes partial residuals for every model to have a mean of zero before smoothing and plotting

Details

For the goodness-of-fit test, the le Cessie-van Houwelingen normal test statistic for the unweighted sum of squared errors (Brier score times n) is used. For an ordinal response variable, the test for predicting the probability that $Y \geq j$ is done separately for all j (except the first). Note that the test statistic can have strange behavior (i.e., it is far too large) if the model has no predictive value.

For most of the values of `type`, you must have specified `x=TRUE, y=TRUE` to `lrm` or `orm`.

There is yet no literature on interpreting score residual plots for the ordinal model. Simulations when proportional odds is satisfied have still shown a U-shaped residual plot. The series of binary model score residuals for all cutoffs of Y seems to better check the assumptions. See the examples.

The li.shepherd residual is a single value per observation on the probability scale and can be useful for examining linearity, checking for outliers, and measuring residual correlation.

Value

a matrix (`type="partial","dfbeta","dfbetas","score"`), test statistic (`type="gof"`), or a vector otherwise. For partial residuals from an ordinal model, the returned object is a 3-way array (rows of X by columns of X by cutoffs of Y), and NAs deleted during the fit are not re-inserted into the residuals. For `score.binary`, nothing is returned.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

References

Landwehr, Pregibon, Shoemaker. JASA 79:61–83, 1984.

le Cessie S, van Houwelingen JC. Biometrics 47:1267–1282, 1991.

Hosmer DW, Hosmer T, Lemeshow S, le Cessie S, Lemeshow S. A comparison of goodness-of-fit tests for the logistic regression model. Stat in Med 16:965–980, 1997.

Copas JB. Applied Statistics 38:71–80, 1989.

Li C, Shepherd BE. Biometrika 99:473-480, 2012.

See Also

`lrm`, `orm`, `naresid`, `which.influence`, `loess`, `supsmu`, `lowess`, `boxplot`, `labcurve`

Examples

```
set.seed(1)
x1 <- runif(200, -1, 1)
x2 <- runif(200, -1, 1)
L  <- x1^2 - .5 + x2
y  <- ifelse(runif(200) <= plogis(L), 1, 0)
f <- lrm(y ~ x1 + x2, x=TRUE, y=TRUE)
resid(f)            #add rows for NAs back to data
resid(f, "score")   #also adds back rows
r <- resid(f, "partial")  #for checking transformations of X's
par(mfrow=c(1,2))
for(i in 1:2) {
  xx <- if(i==1)x1 else x2
  plot(xx, r[,i], xlab=c('x1','x2')[i])
  lines(lowess(xx,r[,i]))
}
resid(f, "partial", pl="loess")  #same as last 3 lines
resid(f, "partial", pl=TRUE) #plots for all columns of X using supsmu
resid(f, "gof")           #global test of goodness of fit
lp1 <- resid(f, "lp1")    #approx. leave-out-1 linear predictors
-2*sum(y*lp1 + log(1-plogis(lp1)))  #approx leave-out-1 deviance
                                    #formula assumes y is binary


# Simulate data from a population proportional odds model
set.seed(1)
n   <- 400
age <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
L <- .05*(age-50) + .03*(blood.pressure-120)
p12 <- plogis(L)    # Pr(Y>=1)
p2  <- plogis(L-1)  # Pr(Y=2)
p   <- cbind(1-p12, p12-p2, p2)   # individual class probabilites
# Cumulative probabilities:
cp  <- matrix(cumsum(t(p)) - rep(0:(n-1), rep(3,n)), byrow=TRUE, ncol=3)
# simulate multinomial with varying probs:
y <- (cp < runif(n)) %*% rep(1,3)
y <- as.vector(y)
# Thanks to Dave Krantz for this trick
f <- lrm(y ~ age + blood.pressure, x=TRUE, y=TRUE)
par(mfrow=c(2,2))
resid(f, 'score.binary',   pl=TRUE)              #plot score residuals
resid(f, 'partial', pl=TRUE)                     #plot partial residuals
resid(f, 'gof')           #test GOF for each level separately


# Show use of Li-Shepherd residuals
f.wrong <- lrm(y ~ blood.pressure, x=TRUE, y=TRUE)
par(mfrow=c(2,1))
# li.shepherd residuals from model without age
plot(age, resid(f.wrong, type="li.shepherd"),
     ylab="li.shepherd residual")
```

```
lines(lowess(age, resid(f.wrong, type="li.shepherd")))
# li.shepherd residuals from model including age
plot(age, resid(f, type="li.shepherd"),
     ylab="li.shepherd residual")
lines(lowess(age, resid(f, type="li.shepherd")))


# Make a series of binary fits and draw 2 partial residual plots
#
f1 <- lrm(y>=1 ~ age + blood.pressure, x=TRUE, y=TRUE)
f2  <- update(f1, y==2 ~.)
par(mfrow=c(2,1))
plot.lrm.partial(f1, f2)


# Simulate data from both a proportional odds and a non-proportional
# odds population model.  Check how 3 kinds of residuals detect
# non-prop. odds
set.seed(71)
n <- 400
x <- rnorm(n)

par(mfrow=c(2,3))
for(j in 1:2) {     # 1: prop.odds   2: non-prop. odds
  if(j==1)
    L <- matrix(c(1.4,.4,-.1,-.5,-.9),
                nrow=n, ncol=5, byrow=TRUE) + x / 2
    else {
  # Slopes and intercepts for cutoffs of 1:5 :
  slopes <- c(.7,.5,.3,.3,0)
  ints   <- c(2.5,1.2,0,-1.2,-2.5)
      L <- matrix(ints,   nrow=n, ncol=5, byrow=TRUE) +
           matrix(slopes, nrow=n, ncol=5, byrow=TRUE) * x
    }
  p <- plogis(L)
  # Cell probabilities
  p <- cbind(1-p[,1],p[,1]-p[,2],p[,2]-p[,3],p[,3]-p[,4],p[,4]-p[,5],p[,5])
  # Cumulative probabilities from left to right
  cp  <- matrix(cumsum(t(p)) - rep(0:(n-1), rep(6,n)), byrow=TRUE, ncol=6)
  y   <- (cp < runif(n)) %*% rep(1,6)


  f <- lrm(y ~ x, x=TRUE, y=TRUE)
  for(cutoff in 1:5) print(lrm(y >= cutoff ~ x)$coef)


  print(resid(f,'gof'))
  resid(f, 'score', pl=TRUE)
  # Note that full ordinal model score residuals exhibit a
  # U-shaped pattern even under prop. odds
  ti <- if(j==2) 'Non-Proportional Odds\nSlopes=.7 .5 .3 .3 0' else
    'True Proportional Odds\nOrdinal Model Score Residuals'
  title(ti)
```

```
  resid(f, 'score.binary', pl=TRUE)
  if(j==1) ti <- 'True Proportional Odds\nBinary Score Residuals'
  title(ti)
  resid(f, 'partial', pl=TRUE)
  if(j==1) ti <- 'True Proportional Odds\nPartial Residuals'
  title(ti)
}
par(mfrow=c(1,1))

# Shepherd-Li residuals from orm.  Thanks: Qi Liu

set.seed(3)
n  <- 100
x1 <- rnorm(n)
y  <- x1 + rnorm(n)
g <- orm(y ~ x1, family='probit', x=TRUE, y=TRUE)
g.resid <- resid(g)
plot(x1, g.resid, cex=0.4); lines(lowess(x1, g.resid)); abline(h=0, col=2,lty=2)

set.seed(3)
n <- 100
x1 <- rnorm(n)
y <- x1 + x1^2 +rnorm(n)
# model misspecification, the square term is left out in the model
g <- orm(y ~ x1, family='probit', x=TRUE, y=TRUE)
g.resid <- resid(g)
plot(x1, g.resid, cex=0.4); lines(lowess(x1, g.resid)); abline(h=0, col=2,lty=2)


## Not run:
# Get data used in Hosmer et al. paper and reproduce their calculations
v <- Cs(id, low, age, lwt, race, smoke, ptl, ht, ui, ftv, bwt)
d <- read.table("http://www.umass.edu/statdata/statdata/data/lowbwt.dat",
                skip=6, col.names=v)
d <- upData(d, race=factor(race,1:3,c('white','black','other')))
f <- lrm(low ~ age + lwt + race + smoke, data=d, x=TRUE,y=TRUE)
f
resid(f, 'gof')
# Their Table 7 Line 2 found sum of squared errors=36.91, expected
# value under H0=36.45, variance=.065, P=.071
# We got 36.90, 36.45, SD=.26055 (var=.068), P=.085
# Note that two logistic regression coefficients differed a bit
# from their Table 1

## End(Not run)
```

`residuals.ols` *Residuals for ols*

Description

Computes various residuals and measures of influence for a fit from `ols`.

Usage

```
## S3 method for class 'ols'
residuals(object,
      type=c("ordinary", "score", "dfbeta", "dfbetas",
             "dffit", "dffits", "hat", "hscore", "influence.measures",
             "studentized"), ...)
```

Arguments

`object`	object created by `ols`. Depending on `type`, you may have had to specify `x=TRUE` to `ols`.
`type`	type of residual desired. `"ordinary"` refers to the usual residual. `"score"` is the matrix of score residuals (contributions to first derivative of log likelihood). `dfbeta` and `dfbetas` mean respectively the raw and normalized matrix of changes in regression coefficients after deleting in turn each observation. The coefficients are normalized by their standard errors. `hat` contains the leverages — diagonals of the "hat" matrix. `dffit` and `dffits` contain respectively the difference and normalized difference in predicted values when each observation is omitted. The S `lm.influence` function is used. When `type="hscore"`, the ordinary residuals are divided by one minus the corresponding hat matrix diagonal element to make residuals have equal variance. When `type="influence.measures"` the model is converted to an `lm` model and `influence.measures(object)$infmat` is returned. This is a matrix with dfbetas for all predictors, dffit, cov.r, Cook's d, and hat. For `type="studentized"` studentized leave-out-one residuals are computed. See the help file for `influence.measures` for more details.
`...`	ignored

Value

a matrix or vector, with places for observations that were originally deleted by `ols` held by `NA`s

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

`lm.influence`, `ols`, `which.influence`

Examples

```
set.seed(1)
x1 <- rnorm(100)
x2 <- rnorm(100)
x1[1] <- 100
```

```
y <- x1 + x2 + rnorm(100)
f <- ols(y ~ x1 + x2, x=TRUE, y=TRUE)
resid(f, "dfbetas")
which.influence(f)
i <- resid(f, 'influence.measures') # dfbeta, dffit, etc.
```

---

rexVar *rexVar*

Description

Relative Explained Variation

Usage

```
rexVar(object, data, ns = 500, cint = 0.95)
```

Arguments

object	a fit from rms or rmsb
data	a data frame, data table, or list providing the predictors used in the original fit
ns	maximum number of bootstrap repetitions or posterior draws to use
cint	confidence interval coverage probability for nonparametric bootstrap percentile intervals, or probability for a Bayesian highest posterior density interval for the relative explained variations.

Details

Computes measures of relative explained variation for each predictor in an rms or rmsb model fit object. This is similar to plot(anova(fit), what='proportion R2'). For an ols model the result is exactly that. Uncertainty intervals are computed if the model fit is from rmsb or was run through bootcov() with coef.reps=TRUE. The results may be printed, and there is also a plot method.

When object is not an ols fit, the linear predictor from the fit in object is predicted from the original predictors, resulting in a linear model with $R^2 = 1.0$. The partial R^2 for each predictor from a new ols fit is the relative explained variation. The process is repeated when bootstrap coefficients repetitions or posterior draws are present, to get uncertainty intervals. So relative explained variation is the proportion of variation in the initial model's predicted values (on the linear predictor scale) that is due to each predictor.

Nonlinear and interaction terms are pooled with main linear effect of predictors, so relative explained variation for a predictor measures its total impact on predicted values, either as main effects or effect modifiers (interaction components).

Value

a vector (if bootstrapping or Bayesian posterior sampling was not done) or a matrix otherwise, with rows corresponding to predictors and colums REV, Lower, Upper. The returned object is of class rexVar.

Author(s)

Frank Harrell

See Also

Hmisc::cutGn()

Examples

```
set.seed(1)
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
yo  <- x1 + x2 + rnorm(n) / 2.
# Minimally group y so that bootstrap samples are very unlikely to miss a
# value of y
y <- ordGroupBoot(yo)
d <- data.frame(x1, x2, x3, y)
dd <- datadist(d); options(datadist='dd')
f  <- ols(y ~ pol(x1, 2) * pol(x2, 2) + x3,
          data=d, x=TRUE, y=TRUE)
plot(anova(f), what='proportion R2', pl=FALSE)
rexVar(f)
g <- bootcov(f, B=20, coef.reps=TRUE)
rexVar(g, data=d)
f <- orm(y ~ pol(x1,2) * pol(x2, 2) + x3,
         data=d, x=TRUE, y=TRUE)
rexVar(f, data=d)
g <- bootcov(f, B=20, coef.reps=TRUE)
rexVar(g, data=d)
## Not run:
require(rmsb)
h <- blrm(y ~ pol(x1,2) * pol(x2, 2) + x3, data=d)
rexVar(h, data=d)

## End(Not run)
options(datadist=NULL)
```

---

rms — *rms Methods and Generic Functions*

Description

This is a series of special transformation functions (asis, pol, lsp, rcs, catg, scored, strat, matrx), fitting functions (e.g., lrm,cph, psm, or ols), and generic analysis functions (anova.rms, summary.rms, Predict, plot.Predict, ggplot.Predict, survplot, fastbw, validate, calibrate, specs.rms, which.influence, latexrms, nomogram, datadist, gendata) that help automate

many analysis steps, e.g. fitting restricted interactions and multiple stratification variables, analysis of variance (with tests of linearity of each factor and pooled tests), plotting effects of variables in the model, estimating and graphing effects of variables that appear non-linearly in the model using e.g. inter-quartile-range hazard ratios, bootstrapping model fits, and constructing nomograms for obtaining predictions manually. Behind the scene is the `Design` function which stores extra attributes. `Design()` is not intended to be called by users. `Design` causes detailed design attributes and descriptions of the distribution of predictors to be stored in an attribute of the `terms` component called `Design`.

`modelData` is a replacement for `model.frame.default` that is much streamlined and prepares data for `Design()`. If a second formula is present, `modelData` ensures that missing data deletions are the same for both formulas, and produces a second model frame for `formula2` as the `data2` attribute of the main returned data frame.

Usage

```
modelData(data=environment(formula), formula, formula2=NULL,
          weights, subset, na.action=na.delete, dotexpand=TRUE,
          callenv=parent.frame(n=2))

Design(mf, formula=NULL, specials=NULL, allow.offset=TRUE, intercept=1)
# not to be called by the user; called by fitting routines
# dist <- datadist(x1,x2,sex,age,race,bp)
# or dist <- datadist(my.data.frame)
# Can omit call to datadist if not using summary.rms, Predict,
# survplot.rms, or if all variable settings are given to them
# options(datadist="dist")
# f <- fitting.function(formula = y ~ rcs(x1,4) + rcs(x2,5) + x1%ia%x2 +
#                       rcs(x1,4)%ia%rcs(x2,5) +
#                       strat(sex)*age + strat(race)*bp)
# See rms.trans for rcs, strat, etc.
# %ia% is restricted interaction - not doubly nonlinear
# for x1 by x2 this uses the simple product only, but pools x1*x2
# effect with nonlinear function for overall tests
# specs(f)
# anova(f)
# summary(f)
# fastbw(f)
# pred <- predict(f, newdata=expand.grid(x1=1:10,x2=3,sex="male",
#                 age=50,race="black"))
# pred <- predict(f, newdata=gendata(f, x1=1:10, x2=3, sex="male"))
# This leaves unspecified variables set to reference values from datadist
# pred.combos <- gendata(f, nobs=10)   # Use X-windows to edit predictor settings
# predict(f, newdata=pred.combos)
# plot(Predict(f, x1))  # or ggplot(...)
# latex(f)
# nomogram(f)
```

Arguments

data	a data frame or calling environment
formula	model formula
formula2	an optional second model formula (see for example ppo in blrm)
weights	a weight variable or expression
subset	a subsetting expression evaluated in the calling frame or data
na.action	NA handling function, ideally one such as na.delete that stores extra information about data omissions
specials	a character vector specifying which function evaluations appearing in formula are "special" in the model.frame sense
dotexpand	set to FALSE to prevent . on right hand side of model formula from expanding into all variables in data; used for cph
callenv	the parent frame that called the fitting function
mf	a model frame
allow.offset	set to TRUE if model fitter allows an offset term
intercept	1 if an ordinary intercept is present, 0 otherwise

Value

a data frame augmented with additional information about the predictors and model formulation

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

rms.trans, rmsMisc, cph, lrm, ols, specs.rms, anova.rms, summary.rms, Predict, gendata, fastbw, predictrms. validate, calibrate, which.influence, latex, latexrms, model.frame.default, datadist, describe, nomogram, vif, dataRep

Examples

```
## Not run:
require(rms)
require(ggplot2)
require(survival)
dist <- datadist(data=2)     # can omit if not using summary, (gg)plot, survplot,
                             # or if specify all variable values to them. Can
                             # also  defer.  data=2: get distribution summaries
                             # for all variables in search position 2
                             # run datadist once, for all candidate variables
dist <- datadist(age,race,bp,sex,height)   # alternative
options(datadist="dist")
```

```
f <- cph(Surv(d.time, death) ~ rcs(age,4)*strat(race) +
         bp*strat(sex)+lsp(height,60),x=TRUE,y=TRUE)
anova(f)
anova(f,age,height)          # Joint test of 2 vars
fastbw(f)
summary(f, sex="female")     # Adjust sex to "female" when testing
                             # interacting factor bp
bplot(Predict(f, age, height))   # 3-D plot
ggplot(Predict(f, age=10:70, height=60))
latex(f)                     # LaTeX representation of fit


f <- lm(y ~ x)               # Can use with any fitting function that
                             # calls model.frame.default, e.g. lm, glm
specs.rms(f)                 # Use .rms since class(f)="lm"
anova(f)                     # Works since Varcov(f) (=Varcov.lm(f)) works
fastbw(f)
options(datadist=NULL)
f <- ols(y ~ x1*x2)          # Saves enough information to do fastbw, anova
anova(f)                     # Will not do Predict since distributions
fastbw(f)                    # of predictors not saved
plot(f, x1=seq(100,300,by=.5), x2=.5)
                             # all values defined - don't need datadist
dist <- datadist(x1,x2)      # Equivalent to datadist(f)
options(datadist="dist")
plot(f, x1, x2=.5)        # Now you can do plot, summary
plot(nomogram(f, interact=list(x2=c(.2,.7))))

## End(Not run)
```

`rms.trans` *rms Special Transformation Functions*

Description

This is a series of functions (`asis`, `pol`, `lsp`, `rcs`, `catg`, `scored`, `strat`, `matrx`, `gTrans`, and `%ia%`) that set up special attributes (such as knots and nonlinear term indicators) that are carried through to fits (using for example `lrm`,`cph`, `ols`, `psm`). `anova.rms`, `summary.rms`, `Predict`, `survplot`, `fastbw`, `validate`, `specs`, `which.influence`, `nomogram` and `latex.rms` use these attributes to automate certain analyses (e.g., automatic tests of linearity for each predictor are done by `anova.rms`). Many of the functions are called implicitly. Some S functions such as `ns` derive data-dependent transformations that are not always "remembered" when predicted values are later computed, so the predictions may be incorrect. The functions listed here solve that problem when used in the `rms` context.

`asis` is the identity transformation, `pol` is an ordinary (non-orthogonal) polynomial, `rcs` is a linear tail-restricted cubic spline function (natural spline, for which the `rcspline.eval` function generates the design matrix, the presence of system option `rcspc` causes `rcspline.eval` to be invoked with `pc=TRUE`, and the presence of system option `fractied` causes this value to be passed to `rcspline.eval` as the `fractied` argument), `catg` is for a categorical variable, `scored` is for

an ordered categorical variable, `strat` is for a stratification factor in a Cox model, `matrx` is for a matrix predictor, and `%ia%` represents restricted interactions in which products involving nonlinear effects on both variables are not included in the model. `asis`, `catg`, `scored`, `matrx` are seldom invoked explicitly by the user (only to specify `label` or `name`, usually).

`gTrans` is a general multiple-parameter transformation function. It can be used to specify new polynomial bases, smooth relationships with a discontinuity at one or more values of `x`, grouped categorical variables, e.g., a categorical variable with 5 levels where you want to combine two of the levels to spend only 3 degrees of freedom in all but see plots of predicted values where the two combined categories are kept separate but will have equal effect estimates. The first argument to `gTrans` is a regular numeric, character, or factor variable. The next argument is a function that transforms a vector into a matrix. If the basis functions are to include a linear term it is up too the user to include the original `x` as one of the columns. Column names are assigned automaticall, but any column names specified by the user will override the default name. If you want to signal which terms correspond to linear and which correspond to nonlinear effects for the purpose of running `anova.rms`, add an integer vector attribute `nonlinear` to the resulting matrix. This vector specifies the column numbers corresponding to nonlinear effects. The default is to assume a column is a linear effect. The `parms` attribute stored with a `gTrans` result a character vector version of the function, so as to not waste space carrying along any environment information. If you will be using the `latex` method for typesetting the fitted model, you must include a `tex` attribute also in the produced matrix. This must be a function of a single character string argument (that will ultimately contain the name of the predictor in LaTeX notation) and must produce a vector of LaTeX character strings. See https://hbiostat.org/R/examples/gTrans/gTrans.html for several examples of the use of `gTrans` including the use of `nonlinear` and `tex`.

A `makepredictcall` method is defined so that usage of the transformation functions outside of `rms` fitting functions will work for getting predicted values. Thanks to Therry Therneau for the code.

In the list below, functions `asis` through `gTrans` can have arguments `x`, `parms`, `label`, `name` except that `parms` does not apply to `asis`, `matrx`, `strat`.

Usage

```
asis(...)
matrx(...)
pol(...)
lsp(...)
rcs(...)
catg(...)
scored(...)
strat(...)
gTrans(...)
x1 %ia% x2
## S3 method for class 'rms'
makepredictcall(var, call)
```

Arguments

`...` The arguments ... above contain the following.

`x` a predictor variable (or a function of one). If you specify e.g. `pol(pmin(age,10),3)`, a cubic polynomial will be fitted in `pmin(age,10)` (`pmin` is the S vector

element–by–element function). The predictor will be labeled age in the output, and plots with have age in its original units on the axes. If you use a function such as pmin, the predictor is taken as the first argument, and other arguments must be defined in the frame in effect when predicted values, etc., are computed.

parms parameters of transformation (e.g. number or location of knots). For pol the argument is the order of the polynomial, e.g. 2 for quadratic (the usual default). For lsp it is a vector of knot locations (lsp will not estimate knot locations). For rcs it is the number of knots (if scalar), or vector of knot locations (if >2 elements). The default number is the nknots system option if parms is not given. If the number of knots is given, locations are computed for that number of knots. If system option rcspc is TRUE the parms vector has an attribute defining the principal components transformation parameters. For catg, parms is the category labels (not needed if variable is an S category or factor variable). If omitted, catg will use unique(x), or levels(x) if x is a category or a factor. For scored, parms is a vector of unique values of variable (uses unique(x) by default). This is not needed if x is an S ordered variable. For strat, parms is the category labels (not needed if variable is an S category variable). If omitted, will use unique(x), or levels(x) if x is category or factor. parms is not used for matrix.

label label of predictor for plotting (default = "label" attribute or variable name)

name Name to use for predictor in model. Default is name of argument to function.

x1, x2 two continuous variables for which to form a non-doubly-nonlinear interaction

var a model term passed from a (usually non-rms) function

call call object for a model term

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

rcspline.eval, rcspline.restate, rms, cph, lrm, ols, datadist, makepredictcall

Examples

```
## Not run:
options(knots=4, poly.degree=2)
# To get the old behavior of rcspline.eval knot placement (which didnt' handle
# clumping at the lowest or highest value of the predictor very well):
# options(fractied = 1.0)   # see rcspline.eval for details
country <- factor(country.codes)
blood.pressure <- cbind(sbp=systolic.bp, dbp=diastolic.bp)
```

```
fit <- lrm(Y ~ sqrt(x1)*rcs(x2) + rcs(x3,c(5,10,15)) +
       lsp(x4,c(10,20)) + country + blood.pressure + poly(age,2))
# sqrt(x1) is an implicit asis variable, but limits of x1, not sqrt(x1)
#       are used for later plotting and effect estimation
# x2 fitted with restricted cubic spline with 4 default knots
# x3 fitted with r.c.s. with 3 specified knots
# x4 fitted with linear spline with 2 specified knots
# country is an implied catg variable
# blood.pressure is an implied matrx variable
# since poly is not an rms function (pol is), it creates a
#       matrx type variable with no automatic linearity testing
#       or plotting
f1 <- lrm(y ~ rcs(x1) + rcs(x2) + rcs(x1) %ia% rcs(x2))
# %ia% restricts interactions. Here it removes terms nonlinear in
# both x1 and x2
f2 <- lrm(y ~ rcs(x1) + rcs(x2) + x1 %ia% rcs(x2))
# interaction linear in x1
f3 <- lrm(y ~ rcs(x1) + rcs(x2) + x1 %ia% x2)
# simple product interaction (doubly linear)
# Use x1 %ia% x2 instead of x1:x2 because x1 %ia% x2 triggers
# anova to pool x1*x2 term into x1 terms to test total effect
# of x1
#
# Examples of gTrans
#
# Linear relationship with a discontinuity at zero:
ldisc <- function(x) {z <- cbind(x == 0, x); attr(z, 'nonlinear') <- 1; z}
gTrans(x, ldisc)
# Duplicate pol(x, 2):
pol2 <- function(x) {z <- cbind(x, x^2); attr(z, 'nonlinear') <- 2; z}
gTrans(x, pol2)
# Linear spline with a knot at x=10 with the new slope taking effect
# until x=20 and the spline turning flat at that point but with a
# discontinuous vertical shift
# tex is only needed if you will be using latex(fit)
dspl <- function(x) {
  z <- cbind(x, pmax(pmin(x, 20) - 10, 0), x > 20)
  attr(z, 'nonlinear') <- 2:3
  attr(z, 'tex') <- function(x) sprintf(c('%s', '(\min(%s, 20) - 10)_{+}',
                                          '[%s > 20]'), x)
  z }
gTrans(x, dspl)

## End(Not run)
```

---

rmsMisc *Miscellaneous Design Attributes and Utility Functions*

Description

These functions are used internally to anova.rms, fastbw, etc., to retrieve various attributes of a design. These functions allow some fitting functions not in the rms series (e.g,, lm, glm) to be used with rms.Design, fastbw, and similar functions.

For vcov, there are several functions. The method for orm fits is a bit different because the covariance matrix stored in the fit object only deals with the middle intercept. See the intercepts argument for more options. There is a method for lrm that also allows non-default intercept(s) to be selected (default is first).

The oos.loglik function for each type of model implemented computes the -2 log likelihood for out-of-sample data (i.e., data not necessarily used to fit the model) evaluated at the parameter estimates from a model fit. Vectors for the model's linear predictors and response variable must be given. oos.loglik is used primarily by bootcov.

The Getlim function retrieves distribution summaries from the fit or from a datadist object. It handles getting summaries from both sources to fill in characteristics for variables that were not defined during the model fit. Getlimi returns the summary for an individual model variable.

Mean is a generic function that creates an R function that calculates the expected value of the response variable given a fit from rms or rmsb.

The related.predictors function returns a list containing variable numbers that are directly or indirectly related to each predictor. The interactions.containing function returns indexes of interaction effects containing a given predictor. The param.order function returns a vector of logical indicators for whether parameters are associated with certain types of effects (nonlinear, interaction, nonlinear interaction). combineRelatedPredictors creates of list of inter-connected main effects and interations for use with predictrms with type='ccterms' (useful for gIndex).

The Penalty.matrix function builds a default penalty matrix for non-intercept term(s) for use in penalized maximum likelihood estimation. The Penalty.setup function takes a constant or list describing penalty factors for each type of term in the model and generates the proper vector of penalty multipliers for the current model.

logLik.rms returns the maximized log likelihood for the model, whereas AIC.rms returns the AIC. The latter function has an optional argument for computing AIC on a "chi-square" scale (model likelihood ratio chi-square minus twice the regression degrees of freedom. logLik.ols handles the case for ols, just by invoking logLik.lm in the stats package. logLik.Gls is also defined.

nobs.rms returns the number of observations used in the fit.

The lrtest function does likelihood ratio tests for two nested models, from fits that have stats components with "Model L.R." values. For models such as psm, survreg, ols, lm which have scale parameters, it is assumed that scale parameter for the smaller model is fixed at the estimate from the larger model (see the example). For non-rms fits it is assumed that the logLik function will return the correct -2 log likelihood for a model fit.

univarLR takes a multivariable model fit object from rms and re-fits a sequence of models containing one predictor at a time. It prints a table of likelihood ratio chi^2 statistics from these fits.

The Newlabels function is used to override the variable labels in a fit object. Likewise, Newlevels can be used to create a new fit object with levels of categorical predictors changed. These two functions are especially useful when constructing nomograms.

rmsArgs handles ... arguments to functions such as Predict, summary.rms, nomogram so that variables to vary may be specified without values (after an equals sign).

prModFit is the workhorse for the print methods for highest-level rms model fitting functions, handling both regular, html, and LaTeX printing, the latter two resulting in html or LaTeX code written to the console, automatically ready for knitr. The work of printing summary statistics is done by prStats, which uses the Hmisc print.char.matrix function to print overall model statistics if options(prType=) was not set to "latex" or "html". Otherwise it generates customized LaTeX or html code. The LaTeX longtable and epic packages must be in effect to use LaTeX.

reListclean allows one to rename a subset of a named list, ignoring the previous names and not concatenating them as R does. It also removes NULL elements and (by default) elements that are NA, as when an optional named element is fetched that doesn't exist. It has an argument dec whose elements are correspondingly removed, then dec is appended to the result vector.

formatNP is a function to format a vector of numerics. If digits is specified, formatNP will make sure that the formatted representation has digits positions to the right of the decimal place. If lang="latex" it will translate any scientific notation to LaTeX math form. If lang="html" will convert to html. If pvalue=TRUE, it will replace formatted values with "< 0.0001" (if digits=4).

latex.naprint.delete will, if appropriate, use LaTeX to draw a dot chart of frequency of variable NAs related to model fits. html.naprint.delete does the same thing in the RStudio R markdown context, using Hmisc:dotchartp (which uses plotly) for drawing any needed dot chart.

removeFormulaTerms removes one or more terms from a model formula, using strictly character manipulation. This handles problems such as [.terms removing offset() if you subset on anything. The function can also be used to remove the dependent variable(s) from the formula.

Usage

```
## S3 method for class 'rms'
vcov(object, regcoef.only=TRUE, intercepts='all', ...)
## S3 method for class 'cph'
vcov(object, regcoef.only=TRUE, ...)
## S3 method for class 'Glm'
vcov(object, regcoef.only=TRUE, intercepts='all', ...)
## S3 method for class 'Gls'
vcov(object, intercepts='all', ...)
## S3 method for class 'lrm'
vcov(object, regcoef.only=TRUE, intercepts='all', ...)
## S3 method for class 'ols'
vcov(object, regcoef.only=TRUE, ...)
## S3 method for class 'orm'
vcov(object, regcoef.only=TRUE, intercepts='mid', ...)
## S3 method for class 'psm'
vcov(object, regcoef.only=TRUE, ...)

# Given Design attributes and number of intercepts creates R
# format assign list.  atr non.slopes Terms
DesignAssign(atr, non.slopes, Terms)

oos.loglik(fit, ...)

## S3 method for class 'ols'
```

```
oos.loglik(fit, lp, y, ...)
## S3 method for class 'lrm'
oos.loglik(fit, lp, y, ...)
## S3 method for class 'cph'
oos.loglik(fit, lp, y, ...)
## S3 method for class 'psm'
oos.loglik(fit, lp, y, ...)
## S3 method for class 'Glm'
oos.loglik(fit, lp, y, ...)

Getlim(at, allow.null=FALSE, need.all=TRUE)
Getlimi(name, Limval, need.all=TRUE)

related.predictors(at, type=c("all","direct"))
interactions.containing(at, pred)
combineRelatedPredictors(at)
param.order(at, term.order)

Penalty.matrix(at, X)
Penalty.setup(at, penalty)

## S3 method for class 'Gls'
logLik(object, ...)
## S3 method for class 'ols'
logLik(object, ...)
## S3 method for class 'rms'
logLik(object, ...)
## S3 method for class 'rms'
AIC(object, ..., k=2, type=c('loglik', 'chisq'))
## S3 method for class 'rms'
nobs(object, ...)

lrtest(fit1, fit2)
## S3 method for class 'lrtest'
print(x, ...)

univarLR(fit)

Newlabels(fit, ...)
Newlevels(fit, ...)
## S3 method for class 'rms'
Newlabels(fit, labels, ...)
## S3 method for class 'rms'
Newlevels(fit, levels, ...)

prModFit(x, title, w, digits=4, coefs=TRUE, footer=NULL,
         lines.page=40, long=TRUE, needspace, subtitle=NULL, ...)
```

```
prStats(labels, w, lang=c("plain", "latex", "html"))

reListclean(..., dec=NULL, na.rm=TRUE)

formatNP(x, digits=NULL, pvalue=FALSE,
         lang=c("plain", "latex", "html"))

## S3 method for class 'naprint.delete'
latex(object, file="", append=TRUE, ...)

## S3 method for class 'naprint.delete'
html(object, ...)

removeFormulaTerms(form, which=NULL, delete.response=FALSE)
```

Arguments

fit	result of a fitting function
object	result of a fitting function
regcoef.only	For fits such as parametric survival models which have a final row and column of the covariance matrix for a non-regression parameter such as a log(scale) parameter, setting regcoef.only=TRUE causes only the first p rows and columns of the covariance matrix to be returned, where p is the length of object$coef.
intercepts	set to "none" to omit any rows and columns related to intercepts. Set to an integer scalar or vector to include particular intercept elements. Set to 'all' to include all intercepts, or for orm to "mid" to use the default for orm. The default is to use the first for lrm and the median intercept for orm.
at	Design element of a fit
pred	index of a predictor variable (main effect)
fit1, fit2	fit objects from lrm,ols,psm,cph etc. It doesn't matter which fit object is the sub-model.
lp	linear predictor vector for oos.loglik. For proportional odds ordinal logistic models, this should have used the first intercept only. If lp and y are omitted, the -2 log likelihood for the original fit are returned.
y	values of a new vector of responses passed to oos.loglik.
name	the name of a variable in the model
Limval	an object returned by Getlim
allow.null	prevents Getlim from issuing an error message if no limits are found in the fit or in the object pointed to by options(datadist=)
need.all	set to FALSE to prevent Getlim or Getlimi from issuing an error message if data for a variable are not found
type	For related.predictors, set to "direct" to return lists of indexes of directly related factors only (those in interactions with the predictor). For AIC.rms, type

specifies the basis on which to return AIC. The default is minus twice the maximized log likelihood plus k times the degrees of freedom counting intercept(s). Specify type='chisq' to get a penalized model likelihood ratio chi-square instead.

term.order — 1 for all parameters, 2 for all parameters associated with either nonlinear or interaction effects, 3 for nonlinear effects (main or interaction), 4 for interaction effects, 5 for nonlinear interaction effects.

X — a design matrix, not including columns for intercepts

penalty — a vector or list specifying penalty multipliers for types of model terms

k — the multiplier of the degrees of freedom to be used in computing AIC. The default is 2.

x — a result of lrtest, or the result of a high-level model fitting function (for prModFit)

labels — a character vector specifying new labels for variables in a fit. To give new labels for all variables, you can specify labels of the form labels=c("Age in Years","Cholesterol"), where the list of new labels is assumed to be the length of all main effect-type variables in the fit and in their original order in the model formula. You may specify a named vector to give new labels in random order or for a subset of the variables, e.g., labels=c(age="Age in Years",chol="Cholesterol"). For prStats, is a list with major column headings, which can themselves be vectors that are then stacked vertically.

levels — a list of named vectors specifying new level labels for categorical predictors. This will override parms as well as datadist information (if available) that were stored with the fit.

title — a single character string used to specify an overall title for the regression fit, which is printed first by prModFit. Set to "" to suppress the title.

w — For prModFit, a special list of lists, which each list element specifying information about a block of information to include in the print. output for a fit. For prStats, w is a list of statistics to print, elements of which can be vectors that are stacked vertically. Unnamed elements specify number of digits to the right of the decimal place to which to round (NA means use format without rounding, as with integers and floating point values). Negative values of digits indicate that the value is a P-value to be formatted with formatNP. Digits are recycled as needed.

digits — number of digits to the right of the decimal point, for formatting numeric values in printed output

coefs — specify coefs=FALSE to suppress printing the table of model coefficients, standard errors, etc. Specify coefs=n to print only the first n regression coefficients in the model.

footer — a character string to appear at the bottom of the regression model output

file — name of file to which to write model output

append — specify append=FALSE when using file and you want to start over instead of adding to an existing file.

lang — specifies the typesetting language: plain text, LaTeX, or html

lines.page	see latex
long	set to FALSE to suppress printing of formula and certain other model output
needspace	optional character string to insert inside a LaTeX needspace macro call before the statistics table and before the coefficient matrix, to avoid bad page splits. This assumes the LaTeX needspace style is available. Example: needspace='6\baselineskip' or needspace='1.5in'.
subtitle	optional vector of character strings containing subtitles that will appear under title but not bolded
dec	vector of decimal places used for rounding
na.rm	set to FALSE to keep NAs in the vector created by reListclean
pvalue	set to TRUE if you want values below 10 to the minus digits to be formatted to be less than that value
form	a formula object
which	a vector of one or more character strings specifying the names of functions that are called from a formula, e.g., "cluster". By default no right-hand-side terms are removed.
delete.response	set to TRUE to remove the dependent variable(s) from the formula
atr, non.slopes, Terms	Design function attributes, number of intercepts, and terms object
...	other arguments. For reListclean this contains the elements being extracted. For prModFit this information is passed to the Hmisc latexTabular function when a block of output is a vector to be formatted in LaTeX.

Value

vcov returns a variance-covariance matrix oos.loglik returns a scalar -2 log likelihood value. Getlim returns a list with components limits and values, either stored in fit or retrieved from the object created by datadist and pointed to in options(datadist=). related.predictors and combineRelatedPredictors return a list of vectors, and interactions.containing returns a vector. param.order returns a logical vector corresponding to non-strata terms in the model. Penalty.matrix returns a symmetric matrix with dimension equal to the number of slopes in the model. For all but categorical predictor main effect elements, the matrix is diagonal with values equal to the variances of the columns of X. For segments corresponding to c-1 dummy variables for c-category predictors, puts a c-1 x c-1 sub-matrix in Penalty.matrix that is constructed so that a quadratic form with Penalty.matrix in the middle computes the sum of squared differences in parameter values about the mean, including a portion for the reference cell in which the parameter is by definition zero. Newlabels returns a new fit object with the labels adjusted.

reListclean returns a vector of named (by its arguments) elements. formatNP returns a character vector.

removeFormulaTerms returns a formula object.

See Also

rms, fastbw, anova.rms, summary.lm, summary.glm, datadist, vif, bootcov, latex, latexTabular, latexSN, print.char.matrix,

Examples

```
## Not run:
f <- psm(S ~ x1 + x2 + sex + race, dist='gau')
g <- psm(S ~ x1 + sex + race, dist='gau',
         fixed=list(scale=exp(f$parms)))
lrtest(f, g)


g <- Newlabels(f, c(x2='Label for x2'))
g <- Newlevels(g, list(sex=c('Male','Female'),race=c('B','W')))
nomogram(g)

## End(Not run)
```

`rmsOverview` *Overview of rms Package*

Description

rms is the package that goes along with the book Regression Modeling Strategies. rms does regression modeling, testing, estimation, validation, graphics, prediction, and typesetting by storing enhanced model design attributes in the fit. rms is a re-written version of the Design package that has improved graphics and duplicates very little code in the survival package.

The package is a collection of about 180 functions that assist and streamline modeling, especially for biostatistical and epidemiologic applications. It also contains functions for binary and ordinal logistic regression models and the Buckley-James multiple regression model for right-censored responses, and implements penalized maximum likelihood estimation for logistic and ordinary linear models. rms works with almost any regression model, but it was especially written to work with logistic regression, Cox regression, accelerated failure time models, ordinary linear models, the Buckley-James model, generalized lease squares for longitudinal data (using the nlme package), generalized linear models, and quantile regression (using the quantreg package). rms requires the Hmisc package to be installed. Note that Hmisc has several functions useful for data analysis (especially data reduction and imputation).

Older references below pertaining to the Design package are relevant to rms.

Details

To make use of automatic typesetting features you must have LaTeX or one of its variants installed.

Some aspects of rms (e.g., `latex`) will not work correctly if `options(contrasts=)` other than `c("contr.treatment", "contr.poly")` are used.

rms relies on a wealth of survival analysis functions written by Terry Therneau of Mayo Clinic. Front-ends have been written for several of Therneau's functions, and other functions have been slightly modified.

Statistical Methods Implemented

- Ordinary linear regression models
- Binary and ordinal logistic models (proportional odds and continuation ratio models, probit, log-log, complementary log-log including ordinal cumulative probability models for continuous Y, efficiently handling thousands of distinct Y values using full likelihood methods)
- Bayesian binary and ordinal regression models, partial proportional odds model, and random effects
- Cox model
- Parametric survival models in the accelerated failure time class
- Buckley-James least-squares linear regression model with possibly right-censored responses
- Generalized linear model
- Quantile regression
- Generalized least squares
- Bootstrap model validation to obtain unbiased estimates of model performance without requiring a separate validation sample
- Automatic Wald tests of all effects in the model that are not parameterization-dependent (e.g., tests of nonlinearity of main effects when the variable does not interact with other variables, tests of nonlinearity of interaction effects, tests for whether a predictor is important, either as a main effect or as an effect modifier)
- Graphical depictions of model estimates (effect plots, odds/hazard ratio plots, nomograms that allow model predictions to be obtained manually even when there are nonlinear effects and interactions in the model)
- Various smoothed residual plots, including some new residual plots for verifying ordinal logistic model assumptions
- Composing S functions to evaluate the linear predictor ($X\hat{beta}$), hazard function, survival function, quantile functions analytically from the fitted model
- Typesetting of fitted model using LaTeX
- Robust covariance matrix estimation (Huber or bootstrap)
- Cubic regression splines with linear tail restrictions (natural splines)
- Tensor splines
- Interactions restricted to not be doubly nonlinear
- Penalized maximum likelihood estimation for ordinary linear regression and logistic regression models. Different parts of the model may be penalized by different amounts, e.g., you may want to penalize interaction or nonlinear effects more than main effects or linear effects
- Estimation of hazard or odds ratios in presence of nolinearity and interaction
- Sensitivity analysis for an unmeasured binary confounder in a binary logistic model

Motivation

rms was motivated by the following needs:

- need to automatically print interesting Wald tests that can be constructed from the design

  - tests of linearity with respect to each predictor
  - tests of linearity of interactions
  - pooled interaction tests (e.g., all interactions involving race)
  - pooled tests of effects with higher order effects
    * test of main effect not meaningful when effect in interaction
    * pooled test of main effect + interaction effect is meaningful
    * test of 2nd-order interaction + any 3rd-order interaction containing those factors is meaningful
- need to store transformation parameters with the fit
  - example: knot locations for spline functions
  - these are "remembered" when getting predictions, unlike standard S or R
  - for categorical predictors, save levels so that same dummy variables will be generated for predictions; check that all levels in out-of-data predictions were present when model was fitted
- need for uniform re-insertion of observations deleted because of NAs when using `predict` without `newdata` or when using `resid`
- need to easily plot the regression effect of any predictor
  - example: age is represented by a linear spline with knots at 40 and 60y plot effect of age on log odds of disease, adjusting interacting factors to easily specified constants
  - vary 2 predictors: plot x1 on x-axis, separate curves for discrete x2 or 3d perspective plot for continuous x2
  - if predictor is represented as a function in the model, plots should be with respect to the original variable:
    ```
    f <- lrm(y ~ log(cholesterol)+age)
    plot(Predict(f, cholesterol)) # cholesterol on x-axis, default range
    ggplot(Predict(f, cholesterol)) # same using ggplot2 plotp(Predict(f, cholesterol))
    # same directly using plotly
    ```
- need to store summary of distribution of predictors with the fit
  - plotting limits (default: 10th smallest, 10th largest values or %-tiles)
  - effect limits (default: .25 and .75 quantiles for continuous vars.)
  - adjustment values for other predictors (default: median for continuous predictors, most frequent level for categorical ones)
  - discrete numeric predictors: list of possible values example: x=0,1,2,3,5 -> by default don't plot prediction at x=4
  - values are on the inner-most variable, e.g. cholesterol, not log(chol.)
  - allows estimation/plotting long after original dataset has been deleted
  - for Cox models, underlying survival also stored with fit, so original data not needed to obtain predicted survival curves
- need to automatically print estimates of effects in presence of non- linearity and interaction
  - example: age is quadratic, interacting with sex default effect is inter-quartile-range hazard ratio (for Cox model), for sex=reference level
  - user-controlled effects: `summary(fit, age=c(30,50), sex="female")` -> odds ratios for logistic model, relative survival time for accelerated failure time survival models

  - effects for all variables (e.g. odds ratios) may be plotted with multiple-confidence-level bars
- need for prettier and more concise effect names in printouts, especially for expanded nonlinear terms and interaction terms
  - use inner-most variable name to identify predictors
  - e.g. for `pmin(x^2-3,10)` refer to factor with legal S-name `x`
- need to recognize that an intercept is not always a simple concept
  - some models (e.g., Cox) have no intercept
  - some models (e.g., ordinal logistic) have multiple intercepts
- need for automatic high-quality printing of fitted mathematical model (with dummy variables defined, regression spline terms simplified, interactions "factored"). Focus is on regression splines instead of nonparametric smoothers or smoothing splines, so that explicit formulas for fit may be obtained for use outside S. rms can also compose S functions to evaluate $X\beta$ from the fitted model analytically, as well as compose SAS code to do this.
- need for automatic drawing of nomogram to represent the fitted model
- need for automatic bootstrap validation of a fitted model, with only one S command (with respect to calibration and discrimination)
- need for robust (Huber sandwich) estimator of covariance matrix, and be able to do all other analysis (e.g., plots, C.L.) using the adjusted covariances
- need for robust (bootstrap) estimator of covariance matrix, easily used in other analyses without change
- need for Huber sandwich and bootstrap covariance matrices adjusted for cluster sampling
- need for routine reporting of how many observations were deleted by missing values on each predictor (see `na.delete` in Hmisc)
- need for optional reporting of descriptive statistics for Y stratified by missing status of each X (see na.detail.response)
- need for pretty, annotated survival curves, using the same commands for parametric and Cox models
- need for ordinal logistic model (proportional odds model, continuation ratio model)
- need for estimating and testing general contrasts without having to be conscious of variable coding or parameter order

Fitting Functions Compatible with rms

rms will work with a wide variety of fitting functions, but it is meant especially for the following:

Function	Purpose	Related S Functions
`ols`	Ordinary least squares linear model	`lm`
`lrm`	Binary and ordinal logistic regression model	`glm` `cr.setup`
`orm`	Ordinal regression model	`lrm`
`blrm`	Bayesian binary and ordinal regression	\

psm	Accelerated failure time parametric survival model	survreg
cph	Cox proportional hazards regression	coxph
npsurv	Nonparametric survival estimates	survfit.formula
bj	Buckley-James censored least squares linear model	survreg
Glm	Version of glm for use with rms	glm
Gls	Version of gls for use with rms	gls
Rq	Version of rq for use with rms	rq

Methods in rms

The following generic functions work with fits with rms in effect:

Function	Purpose	Related Functions
print	Print parameters and statistics of fit	
coef	Fitted regression coefficients	
formula	Formula used in the fit	
specs	Detailed specifications of fit	
robcov	Robust covariance matrix estimates	
bootcov	Bootstrap covariance matrix estimates	
summary	Summary of effects of predictors	
plot.summary	Plot continuously shaded confidence bars for results of summary	
anova	Wald tests of most meaningful hypotheses	
contrast	General contrasts, C.L., tests	
plot.anova	Depict results of anova graphically	dotchart
Predict	Partial predictor effects	predict
plot.Predict	Plot predictor effects using lattice graphics	predict
ggplot	Similar to above but using ggplot2	
plotp	Similar to above but using plotly	
bplot	3-D plot of effects of varying two continuous predictors	image, persp, contour
gendata	Generate data frame with predictor combinations (optionally interactively)	expand.grid
predict	Obtain predicted values or design matrix	
fastbw	Fast backward step-down variable selection	step
residuals (or resid)	Residuals, influence statistics from fit	
which.influence	Which observations are overly influential	residuals
sensuc	Sensitivity of one binary predictor in lrm and cph models to an unmeasured binary confounder	
latex	LaTeX representation of fitted	

	model or anova or summary table	
Function	S function analytic representation of a fitted regression model ($X\beta$)	Function.transcan
hazard	S function analytic representation of a fitted hazard function (for psm)	rcspline.restate
Survival	S function analytic representation of fitted survival function (for psm,cph)	
Quantile	S function analytic representation of fitted function for quantiles of survival time (for psm, cph)	
nomogram	Draws a nomogram for the fitted model	latex, plot, ggplot, plotp
survest	Estimate survival probabilities (for psm, cph)	survfit
survplot	Plot survival curves (psm, cph, npsurv)	plot.survfit
validate	Validate indexes of model fit using resampling	val.prob
calibrate	Estimate calibration curve for model using resampling	
vif	Variance inflation factors for a fit	
naresid	Bring elements corresponding to missing data back into predictions and residuals	
naprint	Print summary of missing values	
pentrace	Find optimum penality for penalized MLE	
effective.df	Print effective d.f. for each type of variable in model, for penalized fit or pentrace result	
rm.impute	Impute repeated measures data with non-random dropout *experimental, non-functional*	transcan, fit.mult.impute

Background for Examples

The following programs demonstrate how the pieces of the rms package work together. A (usually) one-time call to the function datadist requires a pass at the entire data frame to store distribution summaries for potential predictor variables. These summaries contain (by default) the .25 and .75 quantiles of continuous variables (for estimating effects such as odds ratios), the 10th smallest and 10th largest values (or .1 and .9 quantiles for small n) for plotting ranges for estimated curves, and the total range. For discrete numeric variables (those having ≤ 10 unique values), the list of unique values is also stored. Such summaries are used by the summary.rms, Predict, and nomogram.rms functions. You may save time and defer running datadist. In that case, the distribution summary is not stored with the fit object, but it can be gathered before running summary, plot, ggplot, or plotp.

```
d <- datadist(my.data.frame) # or datadist(x1,x2)
options(datadist="d") # omit this or use options(datadist=NULL)
# if not run datadist yet
cf <- ols(y ~ x1 * x2)
anova(f)
fastbw(f)
```

`Predict(f, x2)` `predict(f, newdata)`

In the **Examples** section there are three detailed examples using a fitting function designed to be used with rms, `lrm` (logistic regression model). In **Detailed Example 1** we create 3 predictor variables and a two binary response on 500 subjects. For the first binary response, `dz`, the true model involves only `sex` and `age`, and there is a nonlinear interaction between the two because the log odds is a truncated linear relationship in `age` for females and a quadratic function for males. For the second binary outcome, `dz.bp`, the true population model also involves systolic blood pressure (`sys.bp`) through a truncated linear relationship. First, nonparametric estimation of relationships is done using the Hmisc package's `plsmo` function which uses `lowess` with outlier detection turned off for binary responses. Then parametric modeling is done using restricted cubic splines. This modeling does not assume that we know the true transformations for `age` or `sys.bp` but that these transformations are smooth (which is not actually the case in the population).

For **Detailed Example 2**, suppose that a categorical variable treat has values `"a"`, `"b"`, and `"c"`, an ordinal variable `num.diseases` has values 0,1,2,3,4, and that there are two continuous variables, `age` and `cholesterol`. `age` is fitted with a restricted cubic spline, while `cholesterol` is transformed using the transformation `log(cholesterol - 10)`. Cholesterol is missing on three subjects, and we impute these using the overall median cholesterol. We wish to allow for interaction between `treat` and `cholesterol`. The following S program will fit a logistic model, test all effects in the design, estimate effects, and plot estimated transformations. The fit for `num.diseases` really considers the variable to be a 5-level categorical variable. The only difference is that a 3 d.f. test of linearity is done to assess whether the variable can be re-modeled "asis". Here we also show statements to attach the rms package and store predictor characteristics from datadist.

Detailed Example 3 shows some of the survival analysis capabilities of rms related to the Cox proportional hazards model. We simulate data for 2000 subjects with 2 predictors, `age` and `sex`. In the true population model, the log hazard function is linear in `age` and there is no `age` $\times$ `sex` interaction. In the analysis below we do not make use of the linearity in age. rms makes use of many of Terry Therneau's survival functions that are builtin to S.

The following is a typical sequence of steps that would be used with rms in conjunction with the Hmisc `transcan` function to do single imputation of all NAs in the predictors (multiple imputation would be better but would be harder to do in the context of bootstrap model validation), fit a model, do backward stepdown to reduce the number of predictors in the model (with all the severe problems this can entail), and use the bootstrap to validate this stepwise model, repeating the variable selection for each re-sample. Here we take a short cut as the imputation is not repeated within the bootstrap.

In what follows we (atypically) have only 3 candidate predictors. In practice be sure to have the validate and calibrate functions operate on a model fit that contains all predictors that were involved in previous analyses that used the response variable. Here the imputation is necessary because backward stepdown would otherwise delete observations missing on any candidate variable.

Note that you would have to define `x1, x2, x3, y` to run the following code.

```
xt <- transcan(~ x1 + x2 + x3, imputed=TRUE)
impute(xt) # imputes any NAs in x1, x2, x3
# Now fit original full model on filled-in data
f <- lrm(y ~ x1 + rcs(x2,4) + x3, x=TRUE, y=TRUE) #x,y allow boot.
fastbw(f)
# derives stepdown model (using default stopping rule)
validate(f, B=100, bw=TRUE) # repeats fastbw 100 times
cal <- calibrate(f, B=100, bw=TRUE) # also repeats fastbw
plot(cal)
```

Common Problems to Avoid

1. Don't have a formula like `y ~ age + age^2`. In S you need to connect related variables using a function which produces a matrix, such as `pol` or `rcs`. This allows effect estimates (e.g., hazard ratios) to be computed as well as multiple d.f. tests of association.
2. Don't use `poly` or `strata` inside formulas used in rms. Use `pol` and `strat` instead.
3. Almost never code your own dummy variables or interaction variables in S. Let S do this automatically. Otherwise, `anova` can't do its job.
4. Almost never transform predictors outside of the model formula, as then plots of predicted values vs. predictor values, and other displays, would not be made on the original scale. Use instead something like `y ~ log(cell.count+1)`, which will allow `cell.count` to appear on x-axes. You can get fancier, e.g., `y ~ rcs(log(cell.count+1),4)` to fit a restricted cubic spline with 4 knots in `log(cell.count+1)`. For more complex transformations do something like
```
f <- function(x) {
... various 'if' statements, etc.
log(pmin(x,50000)+1)
}
fit1 <- lrm(death ~ f(cell.count))
fit2 <- lrm(death ~ rcs(f(cell.count),4))
}
```
5. Don't put `$` inside variable names used in formulas. Either attach data frames or use `data=`.
6. Don't forget to use `datadist`. Try to use it at the top of your program so that all model fits can automatically take advantage if its distributional summaries for the predictors.
7. Don't `validate` or `calibrate` models which were reduced by dropping "insignificant" predictors. Proper bootstrap or cross-validation must repeat any variable selection steps for each re-sample. Therefore, `validate` or `calibrate` models which contain all candidate predictors, and if you must reduce models, specify the option `bw=TRUE` to `validate` or `calibrate`.
8. Dropping of "insignificant" predictors ruins much of the usual statistical inference for regression models (confidence limits, standard errors, P-values, χ^2, ordinary indexes of model performance) and it also results in models which will have worse predictive discrimination.

Accessing the Package

Use `require(rms)`.

Published Applications of rms and Regression Splines

- Spline fits
  1. Spanos A, Harrell FE, Durack DT (1989): Differential diagnosis of acute meningitis: An analysis of the predictive value of initial observations. *JAMA* 2700-2707.
  2. Ohman EM, Armstrong PW, Christenson RH, *et al.* (1996): Cardiac troponin T levels for risk stratification in acute myocardial ischemia. *New Eng J Med* 335:1333-1341.
- Bootstrap calibration curve for a parametric survival model:
  1. Knaus WA, Harrell FE, Fisher CJ, Wagner DP, *et al.* (1993): The clinical evaluation of new drugs for sepsis: A prospective study design based on survival analysis. *JAMA* 270:1233-1241.

- Splines, interactions with splines, algebraic form of fitted model from `latex.rms`
  1. Knaus WA, Harrell FE, Lynn J, et al. (1995): The SUPPORT prognostic model: Objective estimates of survival for seriously ill hospitalized adults. *Annals of Internal Medicine* 122:191-203.
- Splines, odds ratio chart from fitted model with nonlinear and interaction terms, use of `transcan` for imputation
  1. Lee KL, Woodlief LH, Topol EJ, Weaver WD, Betriu A. Col J, Simoons M, Aylward P, Van de Werf F, Califf RM. Predictors of 30-day mortality in the era of reperfusion for acute myocardial infarction: results from an international trial of 41,021 patients. *Circulation* 1995;91:1659-1668.
- Splines, external validation of logistic models, prediction rules using point tables
  1. Steyerberg EW, Hargrove YV, *et al* (2001): Residual mass histology in testicular cancer: development and validation of a clinical prediction rule. *Stat in Med* 2001;20:3847-3859.
  2. van Gorp MJ, Steyerberg EW, *et al* (2003): Clinical prediction rule for 30-day mortality in Bjork-Shiley convexo-concave valve replacement. *J Clinical Epidemiology* 2003;56:1006-1012.
- Model fitting, bootstrap validation, missing value imputation
  1. Krijnen P, van Jaarsveld BC, Steyerberg EW, Man in 't Veld AJ, Schalekamp, MADH, Habbema JDF (1998): A clinical prediction rule for renal artery stenosis. *Annals of Internal Medicine* 129:705-711.
- Model fitting, splines, bootstrap validation, nomograms
  1. Kattan MW, Eastham JA, Stapleton AMF, Wheeler TM, Scardino PT. A preoperative nomogram for disease recurrence following radical prostatectomy for prostate cancer. *J Natl Ca Inst* 1998; 90(10):766-771.
  2. Kattan, MW, Wheeler TM, Scardino PT. A postoperative nomogram for disease recurrence following radical prostatectomy for prostate cancer. *J Clin Oncol* 1999; 17(5):1499-1507
  3. Kattan MW, Zelefsky MJ, Kupelian PA, Scardino PT, Fuks Z, Leibel SA. A pretreatment nomogram for predicting the outcome of three-dimensional conformal radiotherapy in prostate cancer. *J Clin Oncol* 2000; 18(19):3252-3259.
  4. Eastham JA, May R, Robertson JL, Sartor O, Kattan MW. Development of a nomogram which predicts the probability of a positive prostate biopsy in men with an abnormal digital rectal examination and a prostate specific antigen between 0 and 4 ng/ml. *Urology*. (In press).
  5. Kattan MW, Heller G, Brennan MF. A competing-risk nomogram fir sarcoma-specific death following local recurrence. *Stat in Med* 2003; 22; 3515-3525.
- Penalized maximum likelihood estimation, regression splines, web site to get predicted values
  1. Smits M, Dippel DWJ, Steyerberg EW, et al. Predicting intracranial traumatic findings on computed tomography in patients with minor head injury: The CHIP prediction rule. *Ann Int Med* 2007; 146:397-405.
- Nomogram with 2- and 5-year survival probability and median survival time (but watch out for the use of univariable screening)
  1. Clark TG, Stewart ME, Altman DG, Smyth JF. A prognostic model for ovarian cancer. *Br J Cancer* 2001; 85:944-52.

- Comprehensive example of parametric survival modeling with an extensive nomogram, time ratio chart, anova chart, survival curves generated using survplot, bootstrap calibration curve
  1. Teno JM, Harrell FE, Knaus WA, et al. Prediction of survival for older hospitalized patients: The HELP survival model. *J Am Geriatrics Soc* 2000; 48: S16-S24.
- Model fitting, imputation, and several nomograms expressed in tabular form
  1. Hasdai D, Holmes DR, et al. Cardiogenic shock complicating acute myocardial infarction: Predictors of death. *Am Heart J* 1999; 138:21-31.
- Ordinal logistic model with bootstrap calibration plot
  1. Wu AW, Yasui U, Alzola CF *et al*. Predicting functional status outcomes in hospitalized patients aged 80 years and older. *J Am Geriatric Society* 2000; 48:S6-S15.
- Propensity modeling in evaluating medical diagnosis, anova dot chart
  1. Weiss JP, Gruver C, et al. Ordering an echocardiogram for evaluation of left ventricular function: Level of expertise necessary for efficient use. *J Am Soc Echocardiography* 2000; 13:124-130.
- Simulations using rms to study the properties of various modeling strategies
  1. Steyerberg EW, Eijkemans MJC, Habbema JDF. Stepwise selection in small data sets: A simulation study of bias in logistic regression analysis. *J Clin Epi* 1999; 52:935-942.
  2. Steyerberg WE, Eijekans MJC, Harrell FE, Habbema JDF. Prognostic modeling with logistic regression analysis: In search of a sensible strategy in small data sets. *Med Decision Making* 2001; 21:45-56.
- Statistical methods and references related to rms, along with case studies which includes the rms code which produced the analyses
  1. Harrell FE, Lee KL, Mark DB (1996): Multivariable prognostic models: Issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Stat in Med* 15:361-387.
  2. Harrell FE, Margolis PA, Gove S, Mason KE, Mulholland EK et al. (1998): Development of a clinical prediction model for an ordinal outcome: The World Health Organization ARI Multicentre Study of clinical signs and etiologic agents of pneumonia, sepsis, and meningitis in young infants. *Stat in Med* 17:909-944.
  3. Bender R, Benner, A (2000): Calculating ordinal regression models in SAS and S-Plus. *Biometrical J* 42:677-699.

Bug Reports

The author is willing to help with problems. Send E-mail to `<fh@fharrell.com>`. To report bugs, please do the following:

1. If the bug occurs when running a function on a fit object (e.g., `anova`), attach a `dump`'d text version of the fit object to your note. If you used `datadist` but not until after the fit was created, also send the object created by `datadist`. Example: `save(myfit,"/tmp/myfit.rda")` will create an R binary save file that can be attached to the E-mail.
2. If the bug occurs during a model fit (e.g., with `lrm, ols, psm, cph`), send the statement causing the error with a `save`'d version of the data frame used in the fit. If this data frame is very large, reduce it to a small subset which still causes the error.

Copyright Notice

GENERAL DISCLAIMER This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. In short: you may use this code any way you like, as long as you don't charge money for it, remove this notice, or hold anyone liable for its results. Also, please acknowledge the source and communicate changes to the author.

If this software is used is work presented for publication, kindly reference it using for example: Harrell FE (2009): rms: S functions for biostatistical/epidemiologic modeling, testing, estimation, validation, graphics, and prediction. Programs available from https://hbiostat.org/R/rms/. Be sure to reference other packages used as well as R itself.

Author(s)

Frank E Harrell Jr
Professor of Biostatistics
Vanderbilt University School of Medicine
Nashville, Tennessee
<fh@fharrell.com>

References

The primary resource for the rms package is *Regression Modeling Strategies, second edition* by FE Harrell (Springer-Verlag, 2015) and the web page https://hbiostat.org/R/rms/. See also the Statistics in Medicine articles by Harrell *et al* listed below for case studies of modeling and model validation using rms.

Several datasets useful for multivariable modeling with rms are found at https://hbiostat.org/data/.

Examples

```
## To run several comprehensive examples, run the following command
## Not run:
demo(all, 'rms')

## End(Not run)
```

---

robcov *Robust Covariance Matrix Estimates*

Description

Uses the Huber-White method to adjust the variance-covariance matrix of a fit from maximum likelihood or least squares, to correct for heteroscedasticity and for correlated responses from cluster samples. The method uses the ordinary estimates of regression coefficients and other parameters of the model, but involves correcting the covariance matrix for model misspecification and sampling design. Models currently implemented are models that have a `residuals(fit,type="score")` function implemented, such as `lrm`, `cph`, `coxph`, and ordinary linear models (`ols`). The fit must have specified the `x=TRUE` and `y=TRUE` options for certain models. Observations in different clusters are assumed to be independent. For the special case where every cluster contains one observation, the corrected covariance matrix returned is the "sandwich" estimator (see Lin and Wei). This is a consistent estimate of the covariance matrix even if the model is misspecified (e.g. heteroscedasticity, underdispersion, wrong covariate form).

For the special case of ols fits, `robcov` can compute the improved (especially for small samples) Efron estimator that adjusts for natural heterogeneity of residuals (see Long and Ervin (2000) estimator HC3).

Usage

```
robcov(fit, cluster, method=c('huber','efron'))
```

Arguments

`fit`	a fit object from the `rms` series
`cluster`	a variable indicating groupings. `cluster` may be any type of vector (factor, character, integer). NAs are not allowed. Unique values of `cluster` indicate possibly correlated groupings of observations. Note the data used in the fit and stored in `fit$x` and `fit$y` may have had observations containing missing values deleted. It is assumed that if any NAs were removed during the original model fitting, an `naresid` function exists to restore NAs so that the rows of the score matrix coincide with `cluster`. If `cluster` is omitted, it defaults to the integers 1,2,...,n to obtain the "sandwich" robust covariance matrix estimate.
`method`	can set to `"efron"` for ols fits (only). Default is Huber-White estimator of the covariance matrix.

Value

a new fit object with the same class as the original fit, and with the element `orig.var` added. `orig.var` is the covariance matrix of the original fit. Also, the original `var` component is replaced with the new Huberized estimates. A component `clusterInfo` is added to contain elements `name` and `n` holding the name of the `cluster` variable and the number of clusters.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

References

Huber, PJ. Proc Fifth Berkeley Symposium Math Stat 1:221–33, 1967.

White, H. Econometrica 50:1–25, 1982.

Lin, DY, Wei, LJ. JASA 84:1074–8, 1989.

Rogers, W. Stata Technical Bulletin STB-8, p. 15–17, 1992.

Rogers, W. Stata Release 3 Manual, `deff`, `loneway`, `huber`, `hreg`, `hlogit` functions.

Long, JS, Ervin, LH. The American Statistician 54:217–224, 2000.

See Also

`bootcov`, `naresid`, `residuals.cph`, `http://gforge.se/gmisc` interfaces `rms` to the `sandwich` package

Examples

```
# In OLS test against more manual approach
set.seed(1)
n <- 15
x1 <- 1:n
x2 <- sample(1:n)
y <- round(x1 + x2 + 8*rnorm(n))
f <- ols(y ~ x1 + x2, x=TRUE, y=TRUE)
vcov(f)
vcov(robcov(f))
X <- f$x
G <- diag(resid(f)^2)
solve(t(X) %*% X) %*% (t(X) %*% G %*% X) %*% solve(t(X) %*% X)

# Duplicate data and adjust for intra-cluster correlation to see that
# the cluster sandwich estimator completely ignored the duplicates
x1 <- c(x1,x1)
x2 <- c(x2,x2)
y  <- c(y, y)
g <- ols(y ~ x1 + x2, x=TRUE, y=TRUE)
vcov(robcov(g, c(1:n, 1:n)))

# A dataset contains a variable number of observations per subject,
# and all observations are laid out in separate rows. The responses
# represent whether or not a given segment of the coronary arteries
# is occluded. Segments of arteries may not operate independently
# in the same patient.  We assume a "working independence model" to
# get estimates of the coefficients, i.e., that estimates assuming
# independence are reasonably efficient.  The job is then to get
# unbiased estimates of variances and covariances of these estimates.

n.subjects <- 30
ages <- rnorm(n.subjects, 50, 15)
sexes  <- factor(sample(c('female','male'), n.subjects, TRUE))
logit <- (ages-50)/5
prob <- plogis(logit)  # true prob not related to sex
```

```
id <- sample(1:n.subjects, 300, TRUE) # subjects sampled multiple times
table(table(id))  # frequencies of number of obs/subject
age <- ages[id]
sex <- sexes[id]
# In truth, observations within subject are independent:
y   <- ifelse(runif(300) <= prob[id], 1, 0)
f <- lrm(y ~ lsp(age,50)*sex, x=TRUE, y=TRUE)
g <- robcov(f, id)
diag(g$var)/diag(f$var)
# add ,group=w to re-sample from within each level of w
anova(g)            # cluster-adjusted Wald statistics
# fastbw(g)         # cluster-adjusted backward elimination
plot(Predict(g, age=30:70, sex='female'))  # cluster-adjusted confidence bands
# or use ggplot(...)

# Get design effects based on inflation of the variances when compared
# with bootstrap estimates which ignore clustering
g2 <- robcov(f)
diag(g$var)/diag(g2$var)


# Get design effects based on pooled tests of factors in model
anova(g2)[,1] / anova(g)[,1]




# A dataset contains one observation per subject, but there may be
# heteroscedasticity or other model misspecification. Obtain
# the robust sandwich estimator of the covariance matrix.


# f <- ols(y ~ pol(age,3), x=TRUE, y=TRUE)
# f.adj <- robcov(f)
```

Rq *rms Package Interface to quantreg Package*

Description

The Rq function is the rms front-end to the quantreg package's rq function. print and latex methods are also provided, and a fitting function RqFit is defined for use in bootstrapping, etc. Its result is a function definition.

For the print method, format of output is controlled by the user previously running options(prType="lang") where lang is "plain" (the default), "latex", or "html". For the latex method, html will actually be used of options(prType='html'). When using html with Quarto or RMarkdown, results='asis' need not be written in the chunk header.

Usage

```
Rq(formula, tau = 0.5, data=environment(formula),
   subset, weights, na.action=na.delete,
   method = "br", model = FALSE, contrasts = NULL,
   se = "nid", hs = TRUE, x = FALSE, y = FALSE, ...)

## S3 method for class 'Rq'
print(x, digits=4, coefs=TRUE, title, ...)

## S3 method for class 'Rq'
latex(object,
           file = '', append=FALSE,
           which, varnames, columns=65, inline=FALSE, caption=NULL, ...)

## S3 method for class 'Rq'
predict(object, ..., kint=1, se.fit=FALSE)

RqFit(fit, wallow=TRUE, passdots=FALSE)
```

Arguments

formula	model formula
tau	the single quantile to estimate. Unlike rq you cannot estimate more than one quantile at one model fitting.
data, subset, weights, na.action, method, model, contrasts, se, hs	see rq
x	set to TRUE to store the design matrix with the fit. For print is an Rq object.
y	set to TRUE to store the response vector with the fit
...	other arguments passed to one of the rq fitting routines. For latex.Rq these are optional arguments passed to latexrms. Ignored for print.Rq. For predict.Rq this is usually just a newdata argument.
digits	number of significant digits used in formatting results in print.Rq.
coefs	specify coefs=FALSE to suppress printing the table of model coefficients, standard errors, etc. Specify coefs=n to print only the first n regression coefficients in the model.
title	a character string title to be passed to prModFit
object	an object created by Rq
file, append, which, varnames, columns, inline, caption	see latexrms
kint	ignored
se.fit	set to TRUE to obtain standard errors of predicted quantiles
fit	an object created by Rq
wallow	set to TRUE if weights are allowed in the current context.
passdots	set to TRUE if ... may be passed to the fitter

Value

Rq returns a list of class "rms", "lassorq" or "scadrq","Rq", and "rq". RqFit returns a function definition. latex.Rq returns an object of class "latex".

Note

The author and developer of methodology in the quantreg package is Roger Koenker.

Author(s)

Frank Harrell

See Also

rq, prModFit, orm

Examples

```
## Not run:
set.seed(1)
n <- 100
x1 <- rnorm(n)
y <- exp(x1 + rnorm(n)/4)
dd <- datadist(x1); options(datadist='dd')
fq2 <- Rq(y ~ pol(x1,2))
anova(fq2)
fq3 <- Rq(y ~ pol(x1,2), tau=.75)
anova(fq3)
pq2 <- Predict(fq2, x1)
pq3 <- Predict(fq3, x1)
p <- rbind(Median=pq2, Q3=pq3)
plot(p, ~ x1 | .set.)
# For superpositioning, with true curves superimposed
a <- function(x, y, ...) {
 x <- unique(x)
 col <- trellis.par.get('superpose.line')$col
 llines(x, exp(x), col=col[1], lty=2)
 llines(x, exp(x + qnorm(.75)/4), col=col[2], lty=2)
}
plot(p, addpanel=a)

## End(Not run)
```

---

sensuc *Sensitivity to Unmeasured Covariables*

Description

Performs an analysis of the sensitivity of a binary treatment (X) effect to an unmeasured binary confounder (U) for a fitted binary logistic or an unstratified non-time-dependent Cox survival model (the function works well for the former, not so well for the latter). This is done by fitting a sequence of models with separately created U variables added to the original model. The sequence of models is formed by simultaneously varying a and b, where a measures the association between U and X and b measures the association between U and Y, where Y is the outcome of interest. For Cox models, an approximate solution is used by letting Y represent some binary classification of the event/censoring time and the event indicator. For example, Y could be just be the event indicator, ignoring time of the event or censoring, or it could be 1 if a subject failed before one year and 0 otherwise. When for each combination of a and b the vector of binary values U is generated, one of two methods is used to constrain the properties of U. With either method, the overall prevalence of U is constrained to be `prev.u`. With the default method (`or.method="x:u y:u"`), U is sampled so that the $X : U$ odds ratio is a and the $Y : U$ odds ratio is b. With the second method, U is sampled according to the model $logit(U = 1|X, Y) = \alpha + \beta * Y + \gamma * X$, where $\beta = \log(b)$ and $\gamma = \log(a)$ and α is determined so that the prevalence of $U = 1$ is `prev.u`. This second method results in the adjusted odds ratio for $Y : U$ given X being b whereas the default method forces the unconditional (marginal) $Y : U$ odds ratio to be b. Rosenbaum uses the default method.

There is a `plot` method for plotting objects created by `sensuc`. Values of a are placed on the x-axis and observed marginal odds or hazards ratios for U (unadjusted ratios) appear on the y-axis. For Cox models, the hazard ratios will not agree exactly with X:event indicator odds ratios but they sometimes be made close through judicious choice of the `event` function. The default plot uses four symbols which differentiate whether for the a, b combination the effect of X adjusted for U (and for any other covariables that were in the original model fit) is positive (usually meaning an effect ratio greater than 1) and "significant", merely positive, not positive and non significant, or not positive but significant. There is also an option to draw the numeric value of the X effect ratio at the a,b combination along with its Z statistic underneath in smaller letters, and an option to draw the effect ratio in one of four colors depending on the significance of the Z statistic.

Usage

```
# fit <- lrm(formula=y ~ x + other.predictors, x=TRUE, y=TRUE)  #or
# fit <- cph(formula=Surv(event.time,event.indicator) ~ x + other.predictors,
#            x=TRUE, y=TRUE)

sensuc(fit,
       or.xu=seq(1, 6, by = 0.5), or.u=or.xu,
       prev.u=0.5, constrain.binary.sample=TRUE,
       or.method=c("x:u y:u","u|x,y"),
       event=function(y) if(is.matrix(y))y[,ncol(y)] else 1*y)

## S3 method for class 'sensuc'
plot(x,  ylim=c((1+trunc(min(x$effect.u)-.01))/
                  ifelse(type=='numbers',2,1),
                  1+trunc(max(x$effect.u)-.01)),
     xlab='Odds Ratio for X:U',
     ylab=if(x$type=='lrm')'Odds Ratio for Y:U' else
          'Hazard Ratio for Y:U',
```

```
digits=2, cex.effect=.75, cex.z=.6*cex.effect,
delta=diff(par('usr')[3:4])/40,
type=c('symbols','numbers','colors'),
pch=c(15,18,5,0), col=c(2,3,1,4), alpha=.05,
impressive.effect=function(x)x > 1,...)
```

Arguments

`fit` result of `lrm` or `cph` with `x=TRUE, y=TRUE`. The first variable in the right hand side of the model formula must have been the binary X variable, and it may not interact with other predictors.

`x` result of `sensuc`

`or.xu` vector of possible odds ratios measuring the $X : U$ association.

`or.u` vector of possible odds ratios measuring the $Y : U$ association. Default is `or.xu`.

`prev.u` desired prevalence of $U = 1$. Default is 0.5, which is usually a "worst case" for sensitivity analyses.

`constrain.binary.sample`
By default, the binary U values are sampled from the appropriate distributions conditional on Y and X so that the proportions of $U = 1$ in each sample are exactly the desired probabilities, to within the closeness of $n\times$probability to an integer. Specify `constrain.binary.sample=FALSE` to sample from ordinary Bernoulli distributions, to allow proportions of $U = 1$ to reflect sampling fluctuations.

`or.method` see above

`event` a function classifying the response variable into a binary event for the purposes of constraining the association between U and Y. For binary logistic models, `event` is left at its default value, which is the identify function, i.e, the original Y values are taken as the events (no other choice makes any sense here). For Cox models, the default `event` function takes the last column of the `Surv` object stored with the fit. For rare events (high proportion of censored observations), odds ratios approximate hazard ratios, so the default is OK. For other cases, the survival times should be considered (probably in conjunction with the event indicators), although it may not be possible to get a high enough hazard ratio between U and Y by sampling U by temporarily making Y binary. See the last example which is for a 2-column `Surv` object (first column of response variable=event time, second=event indicator). When dichotomizing survival time at a given point, it is advantageous to choose the cutpoint so that not many censored survival times preceed the cutpoint. Note that in fitting Cox models to examine sensitivity to U, the original non-dichotomized failure times are used.

`ylim` y-axis limits for `plot`

`xlab` x-axis label

`ylab` y-axis label

`digits` number of digits to the right of the decimal point for drawing numbers on the plot, for `type="numbers"` or `type="colors"`.

cex.effect	character size for drawing effect ratios
cex.z	character size for drawing Z statistics
delta	decrement in y value used to draw Z values below effect ratios
type	specify "symbols" (the default), "numbers", or "colors" (see above)
pch	4 plotting characters corresponding to positive and significant effects for X, positive and non-significant effects, not positive and not significant, not positive but significant
col	4 colors as for pch
alpha	significance level
impressive.effect	a function of the odds or hazard ratio for X returning TRUE for a positive effect. By default, a positive effect is taken to mean a ratio exceeding one.
...	optional arguments passed to plot

Value

sensuc returns an object of class "sensuc" with the following elements: OR.xu (vector of desired $X : U$ odds ratios or a values), OOR.xu (observed marginal $X : U$ odds ratios), OR.u (desired $Y : U$ odds ratios or b values), effect.x (adjusted odds or hazards ratio for X in a model adjusted for U and all of the other predictors), effect.u (unadjusted $Y : U$ odds or hazards ratios), effect.u.adj (adjusted $Y : U$ odds or hazards ratios), Z (Z-statistics), prev.u (input to sensuc), cond.prev.u (matrix with one row per a,b combination, specifying prevalences of U conditional on Y and X combinations), and type ("lrm" or "cph").

Author(s)

Frank Harrell
Mark Conaway
Department of Biostatistics
Vanderbilt University School of Medicine
fh@fharrell.com, mconaway@virginia.edu

References

Rosenbaum, Paul R (1995): Observational Studies. New York: Springer-Verlag.

Rosenbaum P, Rubin D (1983): Assessing sensitivity to an unobserved binary covariate in an observational study with binary outcome. J Roy Statist Soc B 45:212–218.

Lee WC (2011): Bounding the bias of unmeasured factors with confounding and effect-modifying potentials. Stat in Med 30:1007-1017.

See Also

lrm, cph, sample

Examples

```
set.seed(17)
x <- sample(0:1, 500,TRUE)
y <- sample(0:1, 500,TRUE)
y[1:100] <- x[1:100]  # induce an association between x and y
x2 <- rnorm(500)


f <- lrm(y ~ x + x2, x=TRUE, y=TRUE)


#Note: in absence of U odds ratio for x is exp(2nd coefficient)


g <- sensuc(f, c(1,3))


# Note: If the generated sample of U was typical, the odds ratio for
# x dropped had U been known, where U had an odds ratio
# with x of 3 and an odds ratio with y of 3


plot(g)


# Fit a Cox model and check sensitivity to an unmeasured confounder

# require(survival)
# f <- cph(Surv(d.time,death) ~ treatment + pol(age,2)*sex, x=TRUE, y=TRUE)
# sensuc(f, event=function(y) y[,2] & y[,1] < 365.25 )
# Event = failed, with event time before 1 year
# Note: Analysis uses f$y which is a 2-column Surv object
```

---

setPb *Progress Bar for Simulations*

Description

Depending on prevailing `options(showprogress=)` and availability of the `tcltk` package, sets up a progress bar and creates a function for simple updating of the bar as iterations progress. Setting `options(showprogressbar=FALSE)` or `options(showprogressbar='none')` results in no progress being shown. Setting the option to something other than `"tk"` or `"none"` results in the console being used to show the current iteration number and intended number of iterations, the same as if `tcltk` is not installed. It is not recommended that the `"tk"` be used for simulations requiring fewer than 10 seconds for more than 100 iterations, as the time required to update the pop-up window will be more than the time required to do the simulations. This problem can be solved by specifying, for example, `every=10` to `setPb` or to the function created by `setPb`, or by using `options(showevery=10)` before `setPb` is called. If `options(showprogress=)` is not specified, progress is shown in the console with an iteration counter.

Usage

```
setPb(n, type = c("Monte Carlo Simulation", "Bootstrap",
                  "Cross-Validation"),
         label, usetk = TRUE, onlytk=FALSE, every=1)
```

Arguments

n	maximum number of iterations
type	type of simulation. Used for the progress bar title if tcltk is being used.
label	used to customize the bar label if present, overriding type
usetk	set to FALSE to override, acting as though the tcltk package were not installed
onlytk	set to TRUE to not write to the console even if tcltk is unavailable and showprogressbar is not FALSE or "none"
every	print a message for every every iterations

Value

a function that should be called by the user once per iteration, specifying the iteration number as the sole argument

Author(s)

Frank Harrell

See Also

tkProgressBar, setTkProgressBar

Examples

```
## Not run:
options(showprogress=TRUE)   # same as ='tk'
pb <- setPb(1000)
for(i in 1:1000) {
   pb(i)   # pb(i, every=10) to only show for multiples of 10
   # your calculations
  }
# Force rms functions to do simulations to not report progress
options(showprogress='none')
# For functions that do simulations to use the console instead of pop-up
# Even with tcltk is installed
options(showprogress='console')
pb <- setPb(1000, label='Random Sampling')

## End(Not run)
```

specs.rms *rms Specifications for Models*

Description

Prints the design specifications, e.g., number of parameters for each factor, levels of categorical factors, knot locations in splines, pre-transformations, etc.

Usage

```
specs(fit, ...)
## S3 method for class 'rms'
specs(fit, long=FALSE, ...)

## S3 method for class 'specs.rms'
print(x, ...)
```

Arguments

fit	a fit object created with the rms library in effect
x	an object returned by specs
long	if TRUE, causes the plotting and estimation limits to be printed for each factor
...	ignored

Value

a list containing information about the fit and the predictors as elements

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

rms, rms.trans, latexrms, datadist

Examples

```
set.seed(1)
blood.pressure <- rnorm(200, 120, 15)
dd <- datadist(blood.pressure)
options(datadist='dd')
L <- .03*(blood.pressure-120)
sick <- ifelse(runif(200) <= plogis(L), 1, 0)
f <- lrm(sick ~ rcs(blood.pressure,5))
```

```
specs(f)    # find out where 5 knots are placed
g <- Glm(sick ~ rcs(blood.pressure,5), family=binomial)
specs(g,long=TRUE)
options(datadist=NULL)
```

---

summary.rms *Summary of Effects in Model*

Description

summary.rms forms a summary of the effects of each factor. When summary is used to estimate odds or hazard ratios for continuous variables, it allows the levels of interacting factors to be easily set, as well as allowing the user to choose the interval for the effect. This method of estimating effects allows for nonlinearity in the predictor. Factors requiring multiple parameters are handled, as summary obtains predicted values at the needed points and takes differences. By default, interquartile range effects (odds ratios, hazards ratios, etc.) are printed for continuous factors, and all comparisons with the reference level are made for categorical factors. print.summary.rms prints the results, latex.summary.rms and html.summary.rms typeset the results, and plot.summary.rms plots shaded confidence bars to display the results graphically. The longest confidence bar on each page is labeled with confidence levels (unless this bar has been ignored due to clip). By default, the following confidence levels are all shown: .9, .95, and .99, using blue of different transparencies. The plot method currently ignores bootstrap and Bayesian highest posterior density intervals but approximates intervals based on standard errors. The html method is for use with R Markdown using html.

The print method will call the latex or html method if options(prType=) is set to "latex" or "html". For "latex" printing through print(), the LaTeX table environment is turned off. When using html with Quarto or RMarkdown, results='asis' need not be written in the chunk header.

If usebootcoef=TRUE and the fit was run through bootcov, the confidence intervals are bootstrap nonparametric percentile confidence intervals, basic bootstrap, or BCa intervals, obtained on contrasts evaluated on all bootstrap samples.

If options(grType='plotly') is in effect and the plotly package is installed, plot is used instead of base graphics to draw the point estimates and confidence limits when the plot method for summary is called. Colors and other graphical arguments to plot.summary are ignored in this case. Various special effects are implemented such as only drawing 0.95 confidence limits by default but including a legend that allows the other CLs to be activated. Hovering over point estimates shows adjustment values if there are any. nbar is not implemented for plotly.

To get more accurate likelihood profile confidence limits, use the contrast.rms function. An example in its help file shows how to get profile likelihood confidence intervals for coefficients or for any contrast.

Usage

```
## S3 method for class 'rms'
summary(object, ..., ycut=NULL, est.all=TRUE, antilog,
conf.int=.95, abbrev=FALSE, vnames=c("names","labels"),
conf.type=c('individual','simultaneous'),
```

```
usebootcoef=TRUE, boot.type=c("percentile","bca","basic"),
posterior.summary=c('mean', 'median', 'mode'), verbose=FALSE)

## S3 method for class 'summary.rms'
print(x, ..., table.env=FALSE)

## S3 method for class 'summary.rms'
latex(object, title, table.env=TRUE, ...)

## S3 method for class 'summary.rms'
html(object, digits=4, dec=NULL, ...)

## S3 method for class 'summary.rms'
plot(x, at, log=FALSE,
    q=c(0.9, 0.95, 0.99), xlim, nbar, cex=1, nint=10,
    cex.main=1, clip=c(-1e30,1e30), main,
    col=rgb(red=.1,green=.1,blue=.8,alpha=c(.1,.4,.7)),
    col.points=rgb(red=.1,green=.1,blue=.8,alpha=1), pch=17,
    lwd=if(length(q) == 1) 3 else 2 : (length(q) + 1), digits=4,
    declim=4, ...)
```

Arguments

`object` a `rms` fit object. Either `options(datadist)` should have been set before the fit, or `datadist()` and `options(datadist)` run before `summary`. For `latex` is the result of `summary`.

`...` For `summary`, omit list of variables to estimate effects for all predictors. Use a list of variables of the form `age, sex` to estimate using default ranges. Specify `age=50` for example to adjust age to 50 when testing other factors (this will only matter for factors that interact with age). Specify e.g. `age=c(40,60)` to estimate the effect of increasing age from 40 to 60. Specify `age=c(40,50,60)` to let age range from 40 to 60 and be adjusted to 50 when testing other interacting factors. For category factors, a single value specifies the reference cell and the adjustment value. For example, if `treat` has levels `"a"`, `"b"` and `"c"` and `treat="b"` is given to `summary`, treatment `a` will be compared to `b` and `c` will be compared to `b`. Treatment `b` will be used when estimating the effect of other factors. Category variables can have category labels listed (in quotes), or an unquoted number that is a legal level, if all levels are numeric. You need only use the first few letters of each variable name - enough for unique identification. For variables not defined with `datadist`, you must specify 3 values, none of which are `NA`.

Also represents other arguments to pass to `latex`, is ignored for `print` and `plot`.

`ycut` must be specified if the fit is a partial proportional odds model. Specifies the single value of the response variable used to estimate ycut-specific regression effects, e.g., odds ratios

`est.all` Set to `FALSE` to only estimate effects of variables listed. Default is `TRUE`.

`antilog` Set to `FALSE` to suppress printing of anti-logged effects. Default is `TRUE` if the

model was fitted by `lrm` or `cph`. Antilogged effects will be odds ratios for logistic models and hazard ratios for proportional hazards models.

`conf.int` Defaults to `.95` for `95%` confidence intervals of effects.

`abbrev` Set to `TRUE` to use the `abbreviate` function to shorten factor levels for categorical variables in the model.

`vnames` Set to `"labels"` to use variable labels to label effects. Default is `"names"` to use variable names.

`conf.type` The default type of confidence interval computed for a given individual (1 d.f.) contrast is a pointwise confidence interval. Set `conf.type="simultaneous"` to use the `multcomp` package's `glht` and `confint` functions to compute confidence intervals with simultaneous (family-wise) coverage, thus adjusting for multiple comparisons. Contrasts are simultaneous only over groups of intervals computed together.

`usebootcoef` If `fit` was the result of `bootcov` but you want to use the bootstrap covariance matrix instead of the nonparametric percentile, basic, or BCa methods for confidence intervals (which uses all the bootstrap coefficients), specify `usebootcoef=FALSE`.

`boot.type` set to `'bca'` to compute BCa confidence limits or to `'basic'` to use the basic bootstrap. The default is to compute percentile intervals.

`posterior.summary` set to `'mode'` or `'median'` to use the posterior mean/median instead of the mean for point estimates of contrasts

`verbose` set to `TRUE` when `conf.type='simultaneous'` to get output describing scope of simultaneous adjustments

`x` result of `summary`

`title` `title` to pass to `latex`. Default is name of fit object passed to `summary` prefixed with `"summary"`.

`table.env` see `latex`

`digits, dec` for `html.summary.rms`; `digits` is the number of significant digits for printing for effects, standard errors, and confidence limits. It is ignored if `dec` is given. The statistics are rounded to `dec` digits to the right of the decimal point of `dec` is given. `digits` is also the number of significant digits to format numeric hover text and labels for `plotly`.

`declim` number of digits to the right of the decimal point to which to round confidence limits for labeling axes

`at` vector of coordinates at which to put tick mark labels on the main axis. If `log=TRUE`, `at` should be in anti-log units.

`log` Set to `TRUE` to plot on $X\beta$ scale but labeled with anti-logs.

`q` scalar or vector of confidence coefficients to depict

`xlim` X-axis limits for `plot` in units of the linear predictors (log scale if `log=TRUE`). If `at` is specified and `xlim` is omitted, `xlim` is derived from the range of `at`.

`nbar` Sets up plot to leave room for `nbar` horizontal bars. Default is the number of non-interaction factors in the model. Set `nbar` to a larger value to keep too much surrounding space from appearing around horizontal bars. If `nbar` is smaller than the number of bars, the plot is divided into multiple pages with up to `nbar` bars on each page.

cex	cex parameter for factor labels.
nint	Number of tick mark numbers for pretty.
cex.main	cex parameter for main title. Set to 0 to suppress the title.
clip	confidence limits outside the interval c(clip[1], clip[2]) will be ignored, and clip also be respected when computing xlim when xlim is not specified. clip should be in the units of fun(x). If log=TRUE, clip should be in $X\beta$ units.
main	main title. Default is inferred from the model and value of log, e.g., "log Odds Ratio".
col	vector of colors, one per value of q
col.points	color for points estimates
pch	symbol for point estimates. Default is solid triangle.
lwd	line width for confidence intervals, corresponding to q

Value

For summary.rms, a matrix of class summary.rms with rows corresponding to factors in the model and columns containing the low and high values for the effects, the range for the effects, the effect point estimates (difference in predicted values for high and low factor values), the standard error of this effect estimate, and the lower and upper confidence limits. If fit$scale.pred has a second level, two rows appear for each factor, the second corresponding to anti–logged effects. Non–categorical factors are stored first, and effects for any categorical factors are stored at the end of the returned matrix. scale.pred and adjust. adjust is a character string containing levels of adjustment variables, if there are any interactions. Otherwise it is "". latex.summary.rms returns an object of class c("latex","file"). It requires the latex function in Hmisc.

Author(s)

Frank Harrell
Hui Nian
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

datadist, rms, rms.trans, rmsMisc, Misc, pretty, contrast.rms

Examples

```
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
label(age)            <- 'Age'      # label is in Hmisc
label(cholesterol)    <- 'Total Cholesterol'
label(blood.pressure) <- 'Systolic Blood Pressure'
```

```
label(sex)            <- 'Sex'
units(cholesterol)    <- 'mg/dl'   # uses units.default in Hmisc
units(blood.pressure) <- 'mmHg'


# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
  (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male'))
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)


ddist <- datadist(age, blood.pressure, cholesterol, sex)
options(datadist='ddist')


fit <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol,4)))


s <- summary(fit)                # Estimate effects using default ranges
                                 # Gets odds ratio for age=3rd quartile
                                 # compared to 1st quartile
## Not run:
latex(s)                         # Use LaTeX to print nice version
latex(s, file="")                # Just write LaTeX code to console
html(s)                          # html/LaTeX to console for knitr
# Or:
options(prType='latex')
summary(fit)                     # prints with LaTeX, table.env=FALSE
options(prType='html')
summary(fit)                     # prints with html

## End(Not run)
summary(fit, sex='male', age=60) # Specify ref. cell and adjustment val
summary(fit, age=c(50,70))       # Estimate effect of increasing age from
                                 # 50 to 70
s <- summary(fit, age=c(50,60,70))
                                 # Increase age from 50 to 70, adjust to
                                 # 60 when estimating effects of other factors
#Could have omitted datadist if specified 3 values for all non-categorical
#variables (1 value for categorical ones - adjustment level)
plot(s, log=TRUE, at=c(.1,.5,1,1.5,2,4,8))


options(datadist=NULL)
```

---

survest.cph *Cox Survival Estimates*

Description

Compute survival probabilities and optional confidence limits for Cox survival models. If `x=TRUE`, `y=TRUE` were specified to `cph`, confidence limits use the correct formula for any combination of predictors. Otherwise, if `surv=TRUE` was specified to `cph`, confidence limits are based only on standard errors of `log(S(t))` at the mean value of $X\beta$. If the model contained only stratification factors, or if predictions are being requested near the mean of each covariable, this approximation will be accurate. Unless `times` is given, at most one observation may be predicted.

Usage

```
survest(fit, ...)
## S3 method for class 'cph'
survest(fit, newdata, linear.predictors, x, times,
        fun, loglog=FALSE, conf.int=0.95, type, vartype,
        conf.type=c("log", "log-log", "plain", "none"), se.fit=TRUE,
        what=c('survival','parallel'),
        individual=FALSE, ...)
```

Arguments

`fit` a model fit from `cph`

`newdata` a data frame containing predictor variable combinations for which predictions are desired

`linear.predictors` a vector of linear predictor values (centered) for which predictions are desired. If the model is stratified, the "strata" attribute must be attached to this vector (see example).

`x` a design matrix at which to compute estimates, with any strata attached as a "strata" attribute. Only one of `newdata`, `linear.predictors`, or `x` may be specified. If none is specified, but `times` is specified, you will get survival predictions at all subjects' linear predictor and strata values.

`times` a vector of times at which to get predictions. If omitted, predictions are made at all unique failure times in the original input data.

`loglog` set to `TRUE` to make the `log-log` transformation of survival estimates and confidence limits.

`fun` any function to transform the estimates and confidence limits (`loglog` is a special case)

`conf.int` set to `FALSE` or `0` to suppress confidence limits, or e.g. `.95` to cause 0.95 confidence limits to be computed

`type` see `survfit.coxph`

`vartype` see `survfit.coxph`

`conf.type` specifies the basis for computing confidence limits. `"log"` is the default as in the `survival` package.

`se.fit` set to `TRUE` to get standard errors of log predicted survival (no matter what `conf.type` is). If `FALSE`, confidence limits are suppressed.

individual	set to TRUE to have survfit interpret newdata as specifying a covariable path for a single individual (represented by multiple records).
what	Normally use what="survival" to estimate survival probabilities at times that may not correspond to the subjects' own times. what="parallel" assumes that the length of times is the number of subjects (or one), and causes survest to estimate the ith subject's survival probability at the ith value of times (or at the scalar value of times). what="parallel" is used by val.surv for example.
...	unused

Details

The result is passed through naresid if newdata, linear.predictors, and x are not specified, to restore placeholders for NAs.

Value

If times is omitted, returns a list with the elements time, n.risk, n.event, surv, call (calling statement), and optionally std.err, upper, lower, conf.type, conf.int. The estimates in this case correspond to one subject. If times is specified, the returned list has possible components time, surv, std.err, lower, and upper. These will be matrices (except for time) if more than one subject is being predicted, with rows representing subjects and columns representing times. If times has only one time, these are reduced to vectors with the number of elements equal to the number of subjects.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

cph, survfit.cph, survfit.coxph, predictrms, survplot

Examples

```
# Simulate data from a population model in which the log hazard
# function is linear in age and there is no age x sex interaction
# Proportional hazards holds for both variables but we
# unnecessarily stratify on sex to see what happens
require(survival)
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('Male','Female'), n, TRUE))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
dt <- -log(runif(n))/h
```

```
label(dt) <- 'Follow-up Time'
e <- ifelse(dt <= cens,1,0)
dt <- pmin(dt, cens)
units(dt) <- "Year"
dd <- datadist(age, sex)
options(datadist='dd')
Srv <- Surv(dt,e)


f <- cph(Srv ~ age*strat(sex), x=TRUE, y=TRUE) #or surv=T
survest(f, expand.grid(age=c(20,40,60),sex=c("Male","Female")),
    times=c(2,4,6), conf.int=.9)
f <- update(f, surv=TRUE)
lp <- c(0, .5, 1)
f$strata   # check strata names
attr(lp,'strata') <- rep(1,3)  # or rep('sex=Female',3)
survest(f, linear.predictors=lp, times=c(2,4,6))

# Test survest by comparing to survfit.coxph for a more complex model
f <- cph(Srv ~ pol(age,2)*strat(sex), x=TRUE, y=TRUE)
survest(f, data.frame(age=median(age), sex=levels(sex)), times=6)

age2 <- age^2
f2 <- coxph(Srv ~ (age + age2)*strata(sex))
new <- data.frame(age=median(age), age2=median(age)^2, sex='Male')
summary(survfit(f2, new), times=6)
new$sex <- 'Female'
summary(survfit(f2, new), times=6)

options(datadist=NULL)
```

---

survest.orm *Title survest.orm*

Description

Title survest.orm

Usage

```
## S3 method for class 'orm'
survest(
  fit,
  newdata = NULL,
  linear.predictors = NULL,
  x = NULL,
  times = NULL,
  fun,
  loglog = FALSE,
  conf.int = 0.95,
```

```
  what = c("survival", "parallel"),
  ...
)
```

Arguments

`fit`	result of `orm`
`newdata`	data frame defining covariate settings
`linear.predictors`	linear predictor vector using the reference intercept
`x`	design matrix
`times`	times for which estimates are desired; defaults to estimating probabilities of T > t for all uncensored times
`fun`	optional transformation of survival probabilities
`loglog`	set to `TRUE` to use the log-log transformatino
`conf.int`	a number between 0-1 with the default of 0.95; set to 0 to not compute CLs
`what`	specify `what='parallel'` to compute the survival probability at the observed linear predictor and time values, both varying; all possible combinations of these are then not created
`...`	ignored

Value

a data frame with variables `time, surv`. If `conf.int > 0` the data also contains `lower, upper`. The variable `Xrow` indicates the row of the design matrix or the linear predictor element used in getting the current data frame row estimate.

Author(s)

Frank Harrell

Examples

```
# See survest.psm
```

`survest.psm` *Parametric Survival Estimates*

Description

Computes predicted survival probabilities or hazards and optionally confidence limits (for survival only) for parametric survival models fitted with `psm`. If getting predictions for more than one observation, `times` must be specified. For a model without predictors, no input data are specified.

Usage

```
## S3 method for class 'psm'
survest(fit, newdata, linear.predictors, x, times, fun,
        loglog=FALSE, conf.int=0.95,
        what=c("survival","hazard","parallel"), ...)

## S3 method for class 'survest.psm'
print(x, ...)
```

Arguments

`fit`	fit from `psm`
`newdata`, `linear.predictors`, `x`, `times`, `conf.int`	see `survest.cph`. One of `newdata`, `linear.predictors`, `x` must be given. `linear.predictors` includes the intercept. If `times` is omitted, predictions are made at 200 equally spaced points between 0 and the maximum failure/censoring time used to fit the model. `x` can also be a result from `survest.psm`.
`what`	The default is to compute survival probabilities. Set `what="hazard"` or some abbreviation of `"hazard"` to compute hazard rates. `what="parallel"` assumes that the length of `times` is the number of subjects (or one), and causes `survest` to estimate the ith subject's survival probability at the ith value of `times` (or at the scalar value of `times`). `what="parallel"` is used by `val.surv` for example.
`loglog`	set to `TRUE` to transform survival estimates and confidence limits using log-log
`fun`	a function to transform estimates and optional confidence intervals
`...`	unused

Details

Confidence intervals are based on asymptotic normality of the linear predictors. The intervals account for the fact that a scale parameter may have been estimated jointly with beta.

Value

see `survest.cph`. If the model has no predictors, predictions are made with respect to varying time only, and the returned object is of class `"npsurv"` so the survival curve can be plotted with `survplot.npsurv`. If `times` is omitted, the entire survival curve or hazard from `t=0,...,fit$maxtime` is estimated, with increments computed to yield 200 points where `fit$maxtime` is the maximum survival time in the data used in model fitting. Otherwise, the `times` vector controls the time points used.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

psm, survreg, rms, survfit, predictrms, survplot, survreg.distributions

Examples

```
# Simulate data from a proportional hazards population model
require(survival)
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50))
dt <- -log(runif(n))/h
label(dt) <- 'Follow-up Time'
e <- ifelse(dt <= cens,1,0)
dt <- pmin(dt, cens)
units(dt) <- "Year"
S <- Surv(dt,e)

f <- psm(S ~ lsp(age,c(40,70)))
survest(f, data.frame(age=seq(20,80,by=5)), times=2)

#Get predicted survival curve for 40 year old
survest(f, data.frame(age=40))

#Get hazard function for 40 year old
survest(f, data.frame(age=40), what="hazard")$surv #still called surv
```

---

survfit.cph *Cox Predicted Survival*

Description

This is a slightly modified version of Therneau's survfit.coxph function. The difference is that survfit.cph assumes that x=TRUE,y=TRUE were specified to the fit. This assures that the environment in effect at the time of the fit (e.g., automatic knot estimation for spline functions) is the same one used for basing predictions.

Usage

```
## S3 method for class 'cph'
survfit(formula, newdata, se.fit=TRUE, conf.int=0.95,
        individual=FALSE, type=NULL, vartype=NULL,
        conf.type=c('log', "log-log", "plain", "none"), censor=TRUE, id, ...)
```

Arguments

formula — a fit object from cph or coxph see survfit.coxph

newdata, se.fit, conf.int, individual, type, vartype, conf.type, censor, id — see survfit. If individual is TRUE, there must be exactly one Surv object in newdata. This object is used to specify time intervals for time-dependent covariate paths. To get predictions for multiple subjects with time-dependent covariates, specify a vector id which specifies unique hypothetical subjects. The length of id should equal the number of rows in newdata.

... — Not used

Value

see survfit.coxph

See Also

survest.cph

---

survplot — *Plot Survival Curves and Hazard Functions*

Description

Plot estimated survival curves, and for parametric survival models, plot hazard functions. There is an option to print the number of subjects at risk at the start of each time interval for certain models. Curves are automatically labeled at the points of maximum separation (using the labcurve function), and there are many other options for labeling that can be specified with the label.curves parameter. For example, different plotting symbols can be placed at constant x-increments and a legend linking the symbols with category labels can automatically positioned on the most empty portion of the plot.

If the fit is from psm and ggplot=TRUE is specified, a ggplot2 graphic will instead be produced using the survplot.orm function.

For the case of a two stratum analysis by npsurv, survdiffplot plots the difference in two Kaplan-Meier estimates along with approximate confidence bands for the differences, with a reference line at zero. The number of subjects at risk is optionally plotted. This number is taken as the minimum of the number of subjects at risk over the two strata. When conf='diffbands', survdiffplot instead does not make a new plot but adds a shaded polygon to an existing plot, showing the midpoint of two survival estimates plus or minus 1/2 the width of the confidence interval for the difference of two Kaplan-Meier estimates.

survplotp creates an interactive plotly graphic with shaded confidence bands for fits other than from orms. In the two strata case, it draws the 1/2 confidence bands for the difference in two probabilities centered at the midpoint of the probability estimates, so that where the two curves touch this band there is no significant difference (no multiplicity adjustment is made). For the two

strata case, the two individual confidence bands have entries in the legend but are not displayed until the user clicks on the legend.

When code was from running npsurv on a multi-state/competing risk Surv object, survplot plots cumulative incidence curves properly accounting for competing risks. You must specify exactly one state/event cause to plot using the state argument. survplot will not plot multiple states on one graph. This can be accomplished using multiple calls with different values of state and specifying add=TRUE for all but the first call.

Usage

```
survplot(fit, ...)
survplotp(fit, ...)
## S3 method for class 'rms'
survplot(fit, ..., xlim,
         ylim=if(loglog) c(-5, 1.5) else if
                 (what == "survival" & missing(fun)) c(0, 1),
         xlab, ylab, time.inc,
         what=c("survival","hazard"),
         type=c("tsiatis","kaplan-meier"),
         conf.type=c("log","log-log","plain","none"),
         conf.int=FALSE, conf=c("bands","bars"), mylim=NULL,
         add=FALSE, label.curves=TRUE,
         abbrev.label=FALSE, levels.only=FALSE,
         lty, lwd=par("lwd"),
         col=1, col.fill=gray(seq(.95, .75, length=5)),
         adj.subtitle=TRUE, loglog=FALSE, fun,
         n.risk=FALSE, logt=FALSE, dots=FALSE, dotsize=.003,
         grid=NULL, srt.n.risk=0, sep.n.risk=0.056, adj.n.risk=1,
         y.n.risk, cex.n.risk=.6, cex.xlab=par('cex.lab'),
         cex.ylab=cex.xlab, pr=FALSE, ggplot=FALSE)
## S3 method for class 'npsurv'
survplot(fit, xlim,
         ylim, xlab, ylab, time.inc, state=NULL,
         conf=c("bands","bars","diffbands","none"), mylim=NULL,
         add=FALSE, label.curves=TRUE, abbrev.label=FALSE,
         levels.only=FALSE, lty,lwd=par('lwd'),
         col=1, col.fill=gray(seq(.95, .75, length=5)),
         loglog=FALSE, fun, n.risk=FALSE, aehaz=FALSE, times=NULL,
         logt=FALSE, dots=FALSE, dotsize=.003, grid=NULL,
         srt.n.risk=0, sep.n.risk=.056, adj.n.risk=1,
         y.n.risk, cex.n.risk=.6, cex.xlab=par('cex.lab'), cex.ylab=cex.xlab,
         pr=FALSE, ...)
## S3 method for class 'npsurv'
survplotp(fit, xlim, ylim, xlab, ylab, time.inc, state=NULL,
         conf=c("bands", "none"), mylim=NULL, abbrev.label=FALSE,
         col=colorspace::rainbow_hcl,  levels.only=TRUE,
         loglog=FALSE, fun=function(y) y, aehaz=FALSE, times=NULL,
         logt=FALSE, pr=FALSE, ...)
```

```
survdiffplot(fit, order=1:2, fun=function(y) y,
           xlim, ylim, xlab, ylab="Difference in Survival Probability",
           time.inc, conf.int, conf=c("shaded", "bands","diffbands","none"),
           add=FALSE, lty=1, lwd=par('lwd'), col=1,
           n.risk=FALSE, grid=NULL,
           srt.n.risk=0, adj.n.risk=1,
           y.n.risk, cex.n.risk=.6, cex.xlab=par('cex.lab'),
           cex.ylab=cex.xlab, convert=function(f) f)
```

Arguments

`fit` result of fit (`cph`, `psm`, `npsurv`, `survest.psm`). For `survdiffplot`, `fit` must be the result of `npsurv`.

`...` list of factors with names used in model. For fits from `npsurv` these arguments do not appear - all strata are plotted. Otherwise the first factor listed is the factor used to determine different survival curves. Any other factors are used to specify single constants to be adjusted to, when defaults given to fitting routine (through `limits`) are not used. The value given to factors is the original coding of data given to fit, except that for categorical or strata factors the text string levels may be specified. The form of values given to the first factor are none (omit the equal sign to use default range or list of all values if variable is discrete), `"text"` if factor is categorical, `c(value1, value2, ...)`, or a function which returns a vector, such as `seq(low,high,by=increment)`. Only the first factor may have the values omitted. In this case the `Low effect`, `Adjust to`, and `High effect` values will be used from `datadist` if the variable is continuous. For variables not defined to `datadist`, you must specify non-missing constant settings (or a vector of settings for the one displayed variable). Note that since `npsurv` objects do not use the variable list in `...`, you can specify any extra arguments to `labcurve` by adding them at the end of the list of arguments. For `survplotp` `...`(e.g., `height`, `width`) is passed to `plotly::plot_ly`.

`xlim` a vector of two numbers specifiying the x-axis range for follow-up time. Default is `(0,maxtime)` where `maxtime` was the `pretty()`d version of the maximum follow-up time in any stratum, stored in `fit$maxtime`. If `logt=TRUE`, default is `(1, log(maxtime))`.

`ylim` y-axis limits. Default is `c(0,1)` for survival, and `c(-5,1.5)` if `loglog=TRUE`. If `fun` or `loglog=TRUE` are given and `ylim` is not, the limits will be computed from the data. For `what="hazard"`, default limits are computed from the first hazard function plotted.

`xlab` x-axis label. Default is `units` attribute of failure time variable given to `Surv`.

`ylab` y-axis label. Default is `"Survival Probability"` or `"log(-log Survival Probability)"`. If `fun` is given, the default is `""`. For `what="hazard"`, the default is `"Hazard Function"`. For a multi-state/competing risk application the default is `"Cumulative Incidence"`.

`time.inc` time increment for labeling the x-axis and printing numbers at risk. If not specified, the value of `time.inc` stored with the model fit will be used.

`state` the state/event cause to use in plotting if the fit was for a multi-state/competing risk `Surv` object

type — specifies type of estimates, "tsiatis" (the default) or "kaplan-meier". "tsiatis" here corresponds to the Breslow estimator. This is ignored if survival estimates stored with surv=TRUE are being used. For fits from npsurv, this argument is also ignored, since it is specified as an argument to npsurv.

conf.type — specifies the basis for confidence limits. This argument is ignored for fits from npsurv.

conf.int — Default is FALSE. Specify e.g. .95 to plot 0.95 confidence bands. For fits from parametric survival models, or Cox models with x=TRUE and y=TRUE specified to the fit, the exact asymptotic formulas will be used to compute standard errors, and confidence limits are based on log(-log S(t)) if loglog=TRUE. If x=TRUE and y=TRUE were not specified to cph but surv=TRUE was, the standard errors stored for the underlying survival curve(s) will be used. These agree with the former if predictions are requested at the mean value of X beta or if there are only stratification factors in the model. This argument is ignored for fits from npsurv, which must have previously specified confidence interval specifications. For survdiffplot if conf.int is not specified, the level used in the call to npsurv will be used.

conf — "bars" for confidence bars at each time.inc time point. If the fit was from cph(..., surv=TRUE), the time.inc used will be that stored with the fit. Use conf="bands" (the default) for bands using standard errors at each failure time. For npsurv objects only, conf may also be "none", indicating that confidence interval information stored with the npsurv result should be ignored. For npsurv and survdiffplot, conf may be "diffbands" whereby a shaded region is drawn for comparing two curves. The polygon is centered at the midpoint of the two survival estimates and the height of the polygon is 1/2 the width of the approximate conf.int pointwise confidence region. Survival curves not overlapping the shaded area are approximately significantly different at the 1 - conf.int level.

mylim — used to curtail computed ylim. When ylim is not given by the user, the computed limits are expanded to force inclusion of the values specified in mylim.

what — defaults to "survival" to plot survival estimates. Set to "hazard" or an abbreviation to plot the hazard function (for psm fits only). Confidence intervals are not available for what="hazard".

add — set to TRUE to add curves to an existing plot.

label.curves — default is TRUE to use labcurve to label curves where they are farthest apart. Set label.curves to a list to specify options to labcurve, e.g., label.curves=list(method="arrow", cex=.8). These option names may be abbreviated in the usual way arguments are abbreviated. Use for example label.curves=list(keys=1:5) to draw symbols (as in pch=1:5 - see points) on the curves and automatically position a legend in the most empty part of the plot. Set label.curves=FALSE to suppress drawing curve labels. The col, lty, lwd, and type parameters are automatically passed to labcurve, although you can override them here. To distinguish curves by line types and still have labcurve construct a legend, use for example label.curves=list(keys="lines"). The negative value for the plotting symbol will suppress a plotting symbol from being drawn either on the curves or in the legend.

abbrev.label	set to TRUE to abbreviate() curve labels that are plotted
levels.only	set to TRUE to remove variablename= from the start of curve labels.
lty	vector of line types to use for different factor levels. Default is c(1,3,4,5,6,7,...).
lwd	vector of line widths to use for different factor levels. Default is current par setting for lwd.
col	color for curve, default is 1. Specify a vector to assign different colors to different curves. For survplotp, col is a vector of colors corresponding to strata, or a function that will be called to generate such colors.
col.fill	a vector of colors to used in filling confidence bands
adj.subtitle	set to FALSE to suppress plotting subtitle with levels of adjustment factors not plotted. Defaults to TRUE. This argument is ignored for npsurv.
loglog	set to TRUE to plot log(-log Survival) instead of Survival
fun	specifies any function to translate estimates and confidence limits before plotting. If the fit is a multi-state object the default for fun is function(y) 1 - y to draw cumulative incidence curves.
logt	set to TRUE to plot log(t) instead of t on the x-axis
n.risk	set to TRUE to add number of subjects at risk for each curve, using the surv.summary created by cph or using the failure times used in fitting the model if y=TRUE was specified to the fit or if the fit was from npsurv. The numbers are placed at the bottom of the graph unless y.n.risk is given. If the fit is from survest.psm, n.risk does not apply.
srt.n.risk	angle of rotation for leftmost number of subjects at risk (since this number may run into the second or into the y-axis). Default is 0.
adj.n.risk	justification for leftmost number at risk. Default is 1 for right justification. Use 0 for left justification, .5 for centered.
sep.n.risk	multiple of upper y limit - lower y limit for separating lines of text containing number of subjects at risk. Default is .056*(ylim[2]-ylim[1]).
y.n.risk	When n.risk=TRUE, the default is to place numbers of patients at risk above the x-axis. You can specify a y-coordinate for the bottom line of the numbers using y.n.risk. Specify y.n.risk='auto' to place the numbers below the x-axis at a distance of 1/3 of the range of ylim.
cex.n.risk	character size for number of subjects at risk (when n.risk is TRUE)
cex.xlab	cex for x-axis label
cex.ylab	cex for y-axis label
dots	set to TRUE to plot a grid of dots. Will be plotted at every time.inc (see cph) and at survival increments of .1 (if d>.4), .05 (if .2 < d <= .4), or .025 (if d <= .2), where d is the range of survival displayed.
dotsize	size of dots in inches
grid	defaults to NULL (not drawing grid lines). Set to TRUE to plot gray(.8) grid lines, or specify any color.
pr	set to TRUE to print survival curve coordinates used in the plots
ggplot	set to TRUE to use survplot.orm to draw the curves instead, for a psm fit

aehaz	set to TRUE to add number of events and exponential distribution hazard rate estimates in curve labels. For competing risk data the number of events is for the cause of interest, and the hazard rate is the number of events divided by the sum of all failure and censoring times.
times	a numeric vector of times at which to compute cumulative incidence probability estimates to add to curve labels
order	an integer vector of length two specifying the order of groups when computing survival differences. The default of 1:2 indicates that the second group is subtracted from the first. Specify order=2:1 to instead subtract the first from the second. A subtitle indicates what was done.
convert	a function to convert the output of summary.survfitms to pick off the data needed for a single state

Details

survplot will not work for Cox models with time-dependent covariables. Use survest or survfit for that purpose.

There is a set a system option mgp.axis.labels to allow x and y-axes to have differing mgp graphical parameters (see par). This is important when labels for y-axis tick marks are to be written horizontally (par(las=1)), as a larger gap between the labels and the tick marks are needed. You can set the axis-specific 2nd component of mgp using mgp.axis.labels(c(xvalue,yvalue)).

Value

list with components adjust (text string specifying adjustment levels) and curve.labels (vector of text strings corresponding to levels of factor used to distinguish curves). For npsurv, the returned value is the vector of strata labels, or NULL if there are no strata.

Side Effects

plots. If par()$mar[4] < 4, issues par(mar=) to increment mar[4] by 2 if n.risk=TRUE and add=FALSE. The user may want to reset par(mar) in this case to not leave such a wide right margin for plots. You usually would issue par(mar=c(5,4,4,2)+.1).

References

Boers M (2004): Null bar and null zone are better than the error bar to compare group means in graphs. J Clin Epi 57:712-715.

See Also

datadist, rms, cph, psm, survest, predictrms, plot.Predict, ggplot.Predict, units, errbar, survfit, survreg.distributions, labcurve, mgp.axis, par,

Examples

```
# Simulate data from a population model in which the log hazard
# function is linear in age and there is no age x sex interaction
require(survival)
```

```
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('male','female'), n, TRUE))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='female'))
dt <- -log(runif(n))/h
label(dt) <- 'Follow-up Time'
e <- ifelse(dt <= cens,1,0)
dt <- pmin(dt, cens)
units(dt) <- "Year"
dd <- datadist(age, sex)
options(datadist='dd')
S <- Surv(dt,e)

# When age is in the model by itself and we predict at the mean age,
# approximate confidence intervals are ok

f <- cph(S ~ age, surv=TRUE)
survplot(f, age=mean(age), conf.int=.95)
g <- cph(S ~ age, x=TRUE, y=TRUE)
survplot(g, age=mean(age), conf.int=.95, add=TRUE, col='red', conf='bars')

# Repeat for an age far from the mean; not ok
survplot(f, age=75, conf.int=.95)
survplot(g, age=75, conf.int=.95, add=TRUE, col='red', conf='bars')


#Plot stratified survival curves by sex, adj for quadratic age effect
# with age x sex interaction (2 d.f. interaction)

f <- cph(S ~ pol(age,2)*strat(sex), x=TRUE, y=TRUE)
#or f <- psm(S ~ pol(age,2)*sex)
Predict(f, sex, age=c(30,50,70))
survplot(f, sex, n.risk=TRUE, levels.only=TRUE)   #Adjust age to median
survplot(f, sex, logt=TRUE, loglog=TRUE)   #Check for Weibull-ness (linearity)
survplot(f, sex=c("male","female"), age=50)
                                        #Would have worked without datadist
                                        #or with an incomplete datadist
survplot(f, sex, label.curves=list(keys=c(2,0), point.inc=2))
                                        #Identify curves with symbols


survplot(f, sex, label.curves=list(keys=c('m','f')))
                                        #Identify curves with single letters


#Plots by quintiles of age, adjusting sex to male
options(digits=3)
survplot(f, age=quantile(age,(1:4)/5), sex="male")
```

```
#Plot survival Kaplan-Meier survival estimates for males
f <- npsurv(S ~ 1, subset=sex=="male")
survplot(f)


#Plot survival for both sexes and show exponential hazard estimates
f <- npsurv(S ~ sex)
survplot(f, aehaz=TRUE)
#Check for log-normal and log-logistic fits
survplot(f, fun=qnorm, ylab="Inverse Normal Transform")
survplot(f, fun=function(y)log(y/(1-y)), ylab="Logit S(t)")

#Plot the difference between sexes
survdiffplot(f)

#Similar but show half-width of confidence intervals centered
#at average of two survival estimates
#See Boers (2004)
survplot(f, conf='diffbands')

options(datadist=NULL)
## Not run:
#
# Time to progression/death for patients with monoclonal gammopathy
# Competing risk curves (cumulative incidence)
# status variable must be a factor with first level denoting right censoring
m <- upData(mgus1, stop = stop / 365.25, units=c(stop='years'),
            labels=c(stop='Follow-up Time'), subset=start == 0)
f <- npsurv(Surv(stop, event) ~ 1, data=m)

# Use survplot for enhanced displays of cumulative incidence curves for
# competing risks

survplot(f, state='pcm', n.risk=TRUE, xlim=c(0, 20), ylim=c(0, .5), col=2)
survplot(f, state='death', aehaz=TRUE, col=3,
         label.curves=list(keys='lines'))
f <- npsurv(Surv(stop, event) ~ sex, data=m)
survplot(f, state='death', aehaz=TRUE, n.risk=TRUE, conf='diffbands',
         label.curves=list(keys='lines'))

# Plot survival curves estimated from an ordinal semiparametric model
f <- orm(Ocens(y, ifelse(y <= cens, y, Inf)) ~ age)
survplot(f, age=c(30, 50))

## End(Not run)
```

---

survplot.orm *Title Survival Curve Plotting*

Description

Plots predicted survival curves with easy specification of predictor settings, with optional confidence bands. For ‘orm‘ fits these are step functions, and for ‘psm‘ fits they are smooth curves.

Usage

```
## S3 method for class 'orm'
survplot(
  fit,
  ...,
  xlab,
  ylab = "Survival Probability",
  conf.int = FALSE,
  conf = c("bands", "bars"),
  facet = FALSE,
  nrow = NULL,
  alpha = 0.15,
  adj.subtitle = TRUE,
  onlydata = FALSE
)
```

Arguments

`fit`	a fit produced by [orm()]; also works for [psm()] fits
`...`	list of factors with names used in model. The first factor listed is the factor used to determine different survival curves. Any other factors are used to specify single constants to be adjusted to, when defaults given to fitting routine (through ‘limits‘) are not used. The value given to factors is the original coding of data given to fit, except that for categorical factors the text string levels may be specified. The form of values given to the first factor are none (omit the equal sign to use default range or list of all values if variable is discrete), ‘"text"‘ if factor is categorical, ‘c(value1, value2, . . .)‘, or a function which returns a vector, such as ‘seq(low,high,by=increment)‘. Only the first factor may have the values omitted. In this case the ‘Low effect‘, ‘Adjust to‘, and ‘High effect‘ values will be used from ‘datadist‘ if the variable is continuous. For variables not defined to ‘datadist‘, you must specify non-missing constant settings (or a vector of settings for the one displayed variable).
`xlab`	character string label for x-axis; uses the ‘plotmath‘-style ‘yplabel‘ for the ‘y‘ variable stored in the fit if ‘xlab‘ is absent
`ylab`	y-axis label, defaulting to ‘"Survival Probability"‘
`conf.int`	defaults to ‘FALSE‘ (same as specifying ‘0‘); specify a positive value less than 1 to get two-sided confidence intervals utilizing approximate normality of linear predictors
`conf`	not currently used
`facet`	set to ‘TRUE‘ to have the first varying variable appear as a facet instead of as different colored step functions

nrow	when faceting on one varying variable using ‘facet_wrap‘ specifies the number of rows to create
alpha	transparency for confidence bands
adj.subtitle	set to ‘FALSE‘ to not show a caption with the values of non-varying values (adjustment variables)
onlydata	set to ‘TRUE‘ to return the data used in ‘ggplot2‘ plotting instead of the graphics object

Value

if ‘onlydata‘ is left at its default value, a ‘ggplot2‘ graphics object for which additional layers may later be added

Author(s)

Frank Harrell md

See Also

[Hmisc::geom_stepconfint()]

Examples

```
set.seed(1)
d <- expand.grid(x1=c('a', 'b', 'c'), x2=c('A','B'), x3=1:2, irep=1:20)
y <- sample(1:10, nrow(d), TRUE)
dd <- datadist(d); options(datadist='dd')
f <- orm(y ~ x1 + x2 + x3, data=d)

survplot(f, x1='a')
survplot(f, x1='a', conf.int=.95)
survplot(f, x1=c('a','b'), x2='A')
survplot(f, x1=c('a', 'b'), x2='A', conf.int=.95)
survplot(f, x1=c('a','b'), x2='A', facet=TRUE)
survplot(f, x1=c('a','b'), x2='A', facet=TRUE, conf.int=.95)

survplot(f, x1=c('a', 'b'), x2=c('A', 'B'))
survplot(f, x1=c('a', 'b'), x2=c('A', 'B'), conf.int=.95)
survplot(f, x1=c('a', 'b'), x2=c('A', 'B'), facet=TRUE)

survplot(f, x1=c('a', 'b'), x2=c('A', 'B'), x3=1:2)

g <- psm(Surv(y) ~ x1 + x2 + x3, data=d)
survplot(g, x1=c('a','b'), x2=c('A', 'B'), ggplot=TRUE)  # calls survplot.orm
# See https://hbiostat.org/rmsc/parsurv#sec-parsurv-assess
# where nonparametric and parametric estimates are combined into one ggplot
options(datadist=NULL)
```

val.prob *Validate Predicted Probabilities*

Description

The val.prob function is useful for validating predicted probabilities against binary events.

Given a set of predicted probabilities p or predicted log odds logit, and a vector of binary outcomes y that were not used in developing the predictions p or logit, val.prob computes the following indexes and statistics: Somers' D_{xy} rank correlation between p and y [$2(C - .5)$, C=ROC area], Nagelkerke-Cox-Snell-Maddala-Magee R-squared index, Discrimination index D [(Logistic model L.R. χ^2 - 1)/n], L.R. χ^2, its P-value, Unreliability index U, χ^2 with 2 d.f. for testing unreliability (H0: intercept=0, slope=1), its P-value, the quality index Q, Brier score (average squared difference in p and y), Intercept, and Slope, E_{max}=maximum absolute difference in predicted and loess-calibrated probabilities, Eavg, the average in same, E90, the 0.9 quantile of same, the Spiegelhalter Z-test for calibration accuracy, and its two-tailed P-value. If pl=TRUE, plots fitted logistic calibration curve and optionally a smooth nonparametric fit using lowess(p,y,iter=0) and grouped proportions vs. mean predicted probability in group. If the predicted probabilities or logits are constant, the statistics are returned and no plot is made. Eavg, Emax, E90 were from linear logistic calibration before rms 4.5-1.

When group is present, different statistics are computed, different graphs are made, and the object returned by val.prob is different. group specifies a stratification variable. Validations are done separately by levels of group and overall. A print method prints summary statistics and several quantiles of predicted probabilities, and a plot method plots calibration curves with summary statistics superimposed, along with selected quantiles of the predicted probabilities (shown as tick marks on calibration curves). Only the lowess calibration curve is estimated. The statistics computed are the average predicted probability, the observed proportion of events, a 1 d.f. chi-square statistic for testing for overall mis-calibration (i.e., a test of the observed vs. the overall average predicted probability of the event) (ChiSq), and a 2 d.f. chi-square statistic for testing simultaneously that the intercept of a linear logistic calibration curve is zero and the slope is one (ChiSq2), average absolute calibration error (average absolute difference between the lowess-estimated calibration curve and the line of identity, labeled Eavg), Eavg divided by the difference between the 0.95 and 0.05 quantiles of predictive probabilities (Eavg/P90), a "median odds ratio", i.e., the anti-log of the median absolute difference between predicted and calibrated predicted log odds of the event (Med OR), the C-index (ROC area), the Brier quadratic error score (B), a chi-square test of goodness of fit based on the Brier score (B ChiSq), and the Brier score computed on calibrated rather than raw predicted probabilities (B cal). The first chi-square test is a test of overall calibration accuracy ("calibration in the large"), and the second will also detect errors such as slope shrinkage caused by overfitting or regression to the mean. See Cox (1970) for both of these score tests. The goodness of fit test based on the (uncalibrated) Brier score is due to Hilden, Habbema, and Bjerregaard (1978) and is discussed in Spiegelhalter (1986). When group is present you can also specify sampling weights (usually frequencies), to obtained weighted calibration curves.

To get the behavior that results from a grouping variable being present without having a grouping variable, use group=TRUE. In the plot method, calibration curves are drawn and labeled by default where they are maximally separated using the labcurve function. The following parameters do not apply when group is present: pl, smooth, logistic.cal, m, g, cuts, emax.lim, legendloc,

riskdist, mkh, connect.group, connect.smooth. The following parameters apply to the plot method but not to val.prob: xlab, ylab, lim, statloc, cex.

Usage

```
val.prob(p, y, logit, group, weights=rep(1,length(y)), normwt=FALSE,
         pl=TRUE, smooth=TRUE, logistic.cal=TRUE,
         xlab="Predicted Probability", ylab="Actual Probability",
         lim=c(0, 1), m, g, cuts, emax.lim=c(0,1),
         legendloc=lim[1] + c(0.55 * diff(lim), 0.27 * diff(lim)),
         statloc=c(0,0.99), riskdist=c("predicted", "calibrated"),
         cex=.7, mkh=.02,
         connect.group=FALSE, connect.smooth=TRUE, g.group=4,
         evaluate=100, nmin=0)

## S3 method for class 'val.prob'
print(x, ...)

## S3 method for class 'val.prob'
plot(x, xlab="Predicted Probability",
     ylab="Actual Probability",
     lim=c(0,1), statloc=lim, stats=1:12, cex=.5,
     lwd.overall=4, quantiles=c(.05,.95), flag, ...)
```

Arguments

p	predicted probability
y	vector of binary outcomes
logit	predicted log odds of outcome. Specify either p or logit.
group	a grouping variable. If numeric this variable is grouped into g.group quantile groups (default is quartiles). Set group=TRUE to use the group algorithm but with a single stratum for val.prob.
weights	an optional numeric vector of per-observation weights (usually frequencies), used only if group is given.
normwt	set to TRUE to make weights sum to the number of non-missing observations.
pl	TRUE to plot calibration curves and optionally statistics
smooth	plot smooth fit to (p,y) using lowess(p,y,iter=0)
logistic.cal	plot linear logistic calibration fit to (p,y)
xlab	x-axis label, default is "Predicted Probability" for val.prob.
ylab	y-axis label, default is "Actual Probability" for val.prob.
lim	limits for both x and y axes
m	If grouped proportions are desired, minimum no. observations per group
g	If grouped proportions are desired, number of quantile groups
cuts	If grouped proportions are desired, actual cut points for constructing intervals, e.g. c(0,.1,.8,.9,1) or seq(0,1,by=.2)

emax.lim	Vector containing lowest and highest predicted probability over which to compute Emax.
legendloc	If pl=TRUE, list with components x,y or vector c(x,y) for upper left corner of legend for curves and points. Default is c(.55, .27) scaled to lim. Use locator(1) to use the mouse, FALSE to suppress legend.
statloc	D_{xy}, C, R^2, D, U, Q, Brier score, Intercept, Slope, and E_{max} will be added to plot, using statloc as the upper left corner of a box (default is c(0,.9)). You can specify a list or a vector. Use locator(1) for the mouse, FALSE to suppress statistics. This is plotted after the curve legends.
riskdist	Use "calibrated" to plot the relative frequency distribution of calibrated probabilities after dividing into 101 bins from lim[1] to lim[2]. Set to "predicted" (the default as of rms 4.5-1) to use raw assigned risk, FALSE to omit risk distribution. Values are scaled so that highest bar is 0.15*(lim[2]-lim[1]).
cex	Character size for legend or for table of statistics when group is given
mkh	Size of symbols for legend. Default is 0.02 (see par()).
connect.group	Defaults to FALSE to only represent group fractions as triangles. Set to TRUE to also connect with a solid line.
connect.smooth	Defaults to TRUE to draw smoothed estimates using a dashed line. Set to FALSE to instead use dots at individual estimates.
g.group	number of quantile groups to use when group is given and variable is numeric.
evaluate	number of points at which to store the lowess-calibration curve. Default is 100. If there are more than evaluate unique predicted probabilities, evaluate equally-spaced quantiles of the unique predicted probabilities, with linearly interpolated calibrated values, are retained for plotting (and stored in the object returned by val.prob.
nmin	applies when group is given. When nmin > 0, val.prob will not store coordinates of smoothed calibration curves in the outer tails, where there are fewer than nmin raw observations represented in those tails. If for example nmin=50, the plot function will only plot the estimated calibration curve from a to b, where there are 50 subjects with predicted probabilities $< a$ and $> b$. nmin is ignored when computing accuracy statistics.
x	result of val.prob (with group in effect)
...	optional arguments for labcurve (through plot). Commonly used options are col (vector of colors for the strata plus overall) and lty. Ignored for print.
stats	vector of column numbers of statistical indexes to write on plot
lwd.overall	line width for plotting the overall calibration curve
quantiles	a vector listing which quantiles should be indicated on each calibration curve using tick marks. The values in quantiles can be any number of values from the following: .01, .025, .05, .1, .25, .5, .75, .9, .95, .975, .99. By default the 0.05 and 0.95 quantiles are indicated.
flag	a function of the matrix of statistics (rows representing groups) returning a vector of character strings (one value for each group, including "Overall"). plot.val.prob will print this vector of character values to the left of the statistics. The flag function can refer to columns of the matrix used as input to the function by their names given in the description above. The default function returns "*" if either ChiSq2 or B ChiSq is significant at the 0.01 level and " " otherwise.

Details

The 2 d.f. χ^2 test and `Med OR` exclude predicted or calibrated predicted probabilities ≤ 0 to zero or ≥ 1, adjusting the sample size as needed.

Value

`val.prob` without `group` returns a vector with the following named elements: `Dxy`, `R2`, `D`, `D:Chi-sq`, `D:p`, `U`, `U:Chi-sq`, `U:p`, `Q`, `Brier`, `Intercept`, `Slope`, `S:z`, `S:p`, `Emax`. When `group` is present `val.prob` returns an object of class `val.prob` containing a list with summary statistics and calibration curves for all the strata plus `"Overall"`.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Harrell FE, Lee KL, Mark DB (1996): Multivariable prognostic models: Issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. Stat in Med 15:361–387.

Harrell FE, Lee KL (1987): Using logistic calibration to assess the accuracy of probability predictions (Technical Report).

Miller ME, Hui SL, Tierney WM (1991): Validation techniques for logistic regression models. Stat in Med 10:1213–1226.

Stallard N (2009): Simple tests for the external validation of mortality prediction scores. Stat in Med 28:377–388.

Harrell FE, Lee KL (1985): A comparison of the *discrimination* of discriminant analysis and logistic regression under multivariate normality. In Biostatistics: Statistics in Biomedical, Public Health, and Environmental Sciences. The Bernard G. Greenberg Volume, ed. PK Sen. New York: North-Holland, p. 333–343.

Cox DR (1970): The Analysis of Binary Data, 1st edition, section 4.4. London: Methuen.

Spiegelhalter DJ (1986):Probabilistic prediction in patient management. Stat in Med 5:421–433.

Rufibach K (2010):Use of Brier score to assess binary predictions. J Clin Epi 63:938-939

Tjur T (2009):Coefficients of determination in logistic regression models-A new proposal:The coefficient of discrimination. Am Statist 63:366–372.

See Also

`validate.lrm`, `lrm.fit`, `lrm`, `labcurve`, `wtd.stats`, `scat1d`

Examples

```
# Fit logistic model on 100 observations simulated from the actual
# model given by Prob(Y=1 given X1, X2, X3) = 1/(1+exp[-(-1 + 2X1)]),
# where X1 is a random uniform [0,1] variable.  Hence X2 and X3 are
# irrelevant.  After fitting a linear additive model in X1, X2,
# and X3, the coefficients are used to predict Prob(Y=1) on a
# separate sample of 100 observations.  Note that data splitting is
# an inefficient validation method unless n > 20,000.


set.seed(1)
n <- 200
x1 <- runif(n)
x2 <- runif(n)
x3 <- runif(n)
logit <- 2*(x1-.5)
P <- 1/(1+exp(-logit))
y <- ifelse(runif(n)<=P, 1, 0)
d <- data.frame(x1,x2,x3,y)
dd <- datadist(d); options(datadist='dd')
f <- lrm(y ~ x1 + x2 + x3, subset=1:100)
pred.logit <- predict(f, d[101:200,])
phat <- 1/(1+exp(-pred.logit))
val.prob(phat, y[101:200], m=20, cex=.5)  # subgroups of 20 obs.


# Validate predictions more stringently by stratifying on whether
# x1 is above or below the median


v <- val.prob(phat, y[101:200], group=x1[101:200], g.group=2)
v
plot(v)
plot(v, flag=function(stats) ifelse(
  stats[,'ChiSq2'] > qchisq(.95,2) |
  stats[,'B ChiSq'] > qchisq(.95,1), '*', ' ') )
# Stars rows of statistics in plot corresponding to significant
# mis-calibration at the 0.05 level instead of the default, 0.01


plot(val.prob(phat, y[101:200], group=x1[101:200], g.group=2),
              col=1:3) # 3 colors (1 for overall)


# Weighted calibration curves
# plot(val.prob(pred, y, group=age, weights=freqs))
options(datadist=NULL)
```

`val.surv` *Validate Predicted Probabilities Against Observed Survival Times*

Description

The `val.surv` function is useful for validating predicted survival probabilities against right-censored failure times. If `u` is specified, the hazard regression function `hare` in the `polspline` package is used to relate predicted survival probability at time `u` to observed survival times (and censoring indicators) to estimate the actual survival probability at time `u` as a function of the estimated survival probability at that time, `est.surv`. If `est.surv` is not given, `fit` must be specified and the `survest` function is used to obtain the predicted values (using `newdata` if it is given, or using the stored linear predictor values if not). `hare` or `movStats` (when `method="smoothkm"`) is given the sole predictor `fun(est.surv)` where `fun` is given by the user or is inferred from `fit`. `fun` is the function of predicted survival probabilities that one expects to create a linear relationship with the linear predictors.

`hare` uses an adaptive procedure to find a linear spline of `fun(est.surv)` in a model where the log hazard is a linear spline in time t, and cross-products between the two splines are allowed so as to not assume proportional hazards. Thus `hare` assumes that the covariate and time functions are smooth but not much else, if the number of events in the dataset is large enough for obtaining a reliable flexible fit. Or specify `method="smoothkm"` to use the `Hmisc` `movStats` function to compute smoothed (by default using `supsmu`) moving window Kaplan-Meier estimates. This method is more flexible than `hare`.

There are special `print` and `plot` methods when `u` is given. In this case, `val.surv` returns an object of class `"val.survh"`, otherwise it returns an object of class `"val.surv"`.

If `u` is not specified, `val.surv` uses Cox-Snell (1968) residuals on the cumulative probability scale to check on the calibration of a survival model against right-censored failure time data. If the predicted survival probability at time t for a subject having predictors X is $S(t|X)$, this method is based on the fact that the predicted probability of failure before time t, $1 - S(t|X)$, when evaluated at the subject's actual survival time T, has a uniform (0,1) distribution. The quantity $1 - S(T|X)$ is right-censored when T is. By getting one minus the Kaplan-Meier estimate of the distribution of $1 - S(T|X)$ and plotting against the 45 degree line we can check for calibration accuracy. A more stringent assessment can be obtained by stratifying this analysis by an important predictor variable. The theoretical uniform distribution is only an approximation when the survival probabilities are estimates and not population values.

When `censor` is specified to `val.surv`, a different validation is done that is more stringent but that only uses the uncensored failure times. This method is used for type I censoring when the theoretical censoring times are known for subjects having uncensored failure times. Let T, C, and F denote respectively the failure time, censoring time, and cumulative failure time distribution ($1 - S$). The expected value of $F(T|X)$ is 0.5 when T represents the subject's actual failure time. The expected value for an uncensored time is the expected value of $F(T|T \leq C, X) = 0.5F(C|X)$. A smooth plot of $F(T|X) - 0.5F(C|X)$ for uncensored T should be a flat line through $y = 0$ if the model is well calibrated. A smooth plot of $2F(T|X)/F(C|X)$ for uncensored T should be a flat line through $y = 1.0$. The smooth plot is obtained by smoothing the (linear predictor, difference or ratio) pairs.

Note that the Cox-Snell residual plot is not very sensitive to model lack of fit.

Usage

```
val.surv(fit, newdata, S, est.surv,
         method=c('hare', 'smoothkm'),
         censor, u, fun, lim, evaluate=100, pred, maxdim=5, ...)
```

```
## S3 method for class 'val.survh'
print(x, ...)

## S3 method for class 'val.survh'
plot(x, lim, xlab, ylab,
                        riskdist=TRUE, add=FALSE,
                        scat1d.opts=list(nhistSpike=200), ...)

## S3 method for class 'val.surv'
plot(x, group, g.group=4,
     what=c('difference','ratio'),
     type=c('l','b','p'),
     xlab, ylab, xlim, ylim, datadensity=TRUE, ...)
```

Arguments

fit	a fit object created by cph or psm
newdata	a data frame for which val.surv should obtain predicted survival probabilities. If omitted, survival estimates are made for all of the subjects used in fit.
S	an Surv object or an Ocens object
est.surv	a vector of estimated survival probabilities corresponding to times in the first column of S.
method	applies if u is specified and defaults to hare
censor	a vector of censoring times. Only the censoring times for uncensored observations are used.
u	a single numeric follow-up time
fun	a function that transforms survival probabilities into the scale of the linear predictor. If fit is given, and represents either a Cox, Weibull, or exponential fit, fun is automatically set to log(-log(p)).
lim	a 2-vector specifying limits of predicted survival probabilities for obtaining estimated actual probabilities at time u. Default for val.surv is the limits for predictions from datadist, which for large n is the 10th smallest and 10th largest predicted survival probability. For plot.val.survh, the default for lim is the range of the combination of predicted probabilities and calibrated actual probabilities. lim is used for both axes of the calibration plot.
evaluate	the number of evenly spaced points over the range of predicted probabilities. This defines the points at which calibrated predictions are obtained for plotting.
pred	a vector of points at which to evaluate predicted probabilities, overriding lim
maxdim	see hare
x	result of val.surv
xlab	x-axis label. For plot.survh, defaults for xlab and ylab come from u and the units of measurement for the raw survival times.
ylab	y-axis label
riskdist	set to FALSE to not call scat1d to draw the distribution of predicted (uncalibrated) probabilities

add	set to TRUE if adding to an existing plot
scat1d.opts	a list of options to pass to scat1d. By default, the option nhistSpike=200 is passed so that a spike histogram is used if the sample size exceeds 200.
...	When u is given to val.surv, ...represents optional arguments to hare or movStats. It can represent arguments to pass to plot or lines for plot.val.survh. Otherwise, ...contains optional arguments for plsmo or plot. For print.val.survh, ...is ignored.
group	a grouping variable. If numeric this variable is grouped into g.group quantile groups (default is quartiles). group, g.group, what, and type apply when u is not given.
g.group	number of quantile groups to use when group is given and variable is numeric.
what	the quantity to plot when censor was in effect. The default is to show the difference between cumulative probabilities and their expectation given the censoring time. Set what="ratio" to show the ratio instead.
type	Set to the default ("l") to plot the trend line only, "b" to plot both individual subjects ratios and trend lines, or "p" to plot only points.
xlim, ylim	axis limits for plot.val.surv when the censor variable was used.
datadensity	By default, plot.val.surv will show the data density on each curve that is created as a result of censor being present. Set datadensity=FALSE to suppress these tick marks drawn by scat1d.

Value

a list of class "val.surv" or "val.survh". Some plot methods return a ggplot2 object.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Cox DR, Snell EJ (1968):A general definition of residuals (with discussion). JRSSB 30:248–275.

Kooperberg C, Stone C, Truong Y (1995): Hazard regression. JASA 90:78–94.

May M, Royston P, Egger M, Justice AC, Sterne JAC (2004):Development and validation of a prognostic model for survival time data: application to prognosis of HIV positive patients treated with antiretroviral therapy. Stat in Med 23:2375–2398.

Stallard N (2009): Simple tests for th external validation of mortality prediction scores. Stat in Med 28:377–388.

See Also

validate, calibrate, hare, scat1d, cph, psm, groupkm

Examples

```
# Generate failure times from an exponential distribution
require(survival)
set.seed(123)              # so can reproduce results
n <- 1000
age <- 50 + 12*rnorm(n)
sex <- factor(sample(c('Male','Female'), n, rep=TRUE, prob=c(.6, .4)))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
t <- -log(runif(n))/h
units(t) <- 'Year'
label(t) <- 'Time to Event'
ev <- ifelse(t <= cens, 1, 0)
t <- pmin(t, cens)
S <- Surv(t, ev)

# First validate true model used to generate data

# If hare is available, make a smooth calibration plot for 1-year
# survival probability where we predict 1-year survival using the
# known true population survival probability
# In addition, use groupkm to show that grouping predictions into
# intervals and computing Kaplan-Meier estimates is not as accurate.

s1 <- exp(-h*1)
w <- val.surv(est.surv=s1, S=S, u=1,
              fun=function(p)log(-log(p)))
plot(w, lim=c(.85,1), scat1d.opts=list(nhistSpike=200, side=1))
groupkm(s1, S, m=100, u=1, pl=TRUE, add=TRUE)

# Now validate the true model using residuals

w <- val.surv(est.surv=exp(-h*t), S=S)
plot(w)
plot(w, group=sex)  # stratify by sex


# Now fit an exponential model and validate
# Note this is not really a validation as we're using the
# training data here
f <- psm(S ~ age + sex, dist='exponential', y=TRUE)
w <- val.surv(f)
plot(w, group=sex)


# We know the censoring time on every subject, so we can
# compare the predicted Pr[T <= observed T | T>c, X] to
# its expectation 0.5 Pr[T <= C | X] where C = censoring time
# We plot a ratio that should equal one
w <- val.surv(f, censor=cens)
plot(w)
plot(w, group=age, g=3)   # stratify by tertile of age
```

validate *Resampling Validation of a Fitted Model's Indexes of Fit*

Description

The `validate` function when used on an object created by one of the `rms` series does resampling validation of a regression model, with or without backward step-down variable deletion. The `print` method will call the `latex` or `html` method if `options(prType=)` is set to `"latex"` or `"html"`. For `"latex"` printing through `print()`, the LaTeX table environment is turned off. When using html with Quarto or RMarkdown, `results='asis'` need not be written in the chunk header.

See `predab.resample` for information about confidence limits.

Usage

```
# fit <- fitting.function(formula=response ~ terms, x=TRUE, y=TRUE)
validate(fit, method="boot", B=40,
         bw=FALSE, rule="aic", type="residual", sls=0.05, aics=0,
         force=NULL, estimates=TRUE, pr=FALSE, ...)
## S3 method for class 'validate'
print(x, digits=4, B=Inf, ...)
## S3 method for class 'validate'
latex(object, digits=4, B=Inf, file='', append=FALSE,
                         title=first.word(deparse(substitute(x))),
                         caption=NULL, table.env=FALSE,
                         size='normalsize', extracolsize=size, ...)
## S3 method for class 'validate'
html(object, digits=4, B=Inf, caption=NULL, ...)
```

Arguments

`fit`	a fit derived by e.g. `lrm`, `cph`, `psm`, `ols`. The options `x=TRUE` and `y=TRUE` must have been specified.
`method`	may be `"crossvalidation"`, `"boot"` (the default), `".632"`, or `"randomization"`. See `predab.resample` for details. Can abbreviate, e.g. `"cross"`, `"b"`, `".6"`.
`B`	number of repetitions. For `method="crossvalidation"`, is the number of groups of omitted observations. For `print.validate`, `latex.validate`, and `html.validate`, `B` is an upper limit on the number of resamples for which information is printed about which variables were selected in each model re-fit. Specify zero to suppress printing. Default is to print all re-samples.
`bw`	`TRUE` to do fast step-down using the `fastbw` function, for both the overall model and for each repetition. `fastbw` keeps parameters together that represent the same factor.
`rule`	Applies if `bw=TRUE`. `"aic"` to use Akaike's information criterion as a stopping rule (i.e., a factor is deleted if the χ^2 falls below twice its degrees of freedom), or `"p"` to use P-values.

type	"residual" or "individual" - stopping rule is for individual factors or for the residual χ^2 for all variables deleted
sls	significance level for a factor to be kept in a model, or for judging the residual χ^2.
aics	cutoff on AIC when rule="aic".
force	see fastbw
estimates	see print.fastbw
pr	TRUE to print results of each repetition
...	parameters for each specific validate function, and parameters to pass to predab.resample (note especially the group, cluster, amd subset parameters). For latex, optional arguments to latex.default. Ignored for html.validate. For psm, you can pass the maxiter parameter here (passed to survreg.control, default is 15 iterations) as well as a tol parameter for judging matrix singularity in solvet (default is 1e-12) and a rel.tolerance parameter that is passed to survreg.control (default is 1e-5). For print.validate ... is ignored.
x, object	an object produced by one of the validate functions
digits	number of decimal places to print
file	file to write LaTeX output. Default is standard output.
append	set to TRUE to append LaTeX output to an existing file
title, caption, table.env, extracolsize	see latex.default. If table.env is FALSE and caption is given, the character string contained in caption will be placed before the table, centered.
size	size of LaTeX output. Default is 'normalsize'. Must be a defined LaTeX size when prepended by double slash.

Details

It provides bias-corrected indexes that are specific to each type of model. For validate.cph and validate.psm, see validate.lrm, which is similar.
For validate.cph and validate.psm, there is an extra argument dxy, which if TRUE causes the dxy.cens function to be invoked to compute the Somers' D_{xy} rank correlation to be computed at each resample. The values corresponding to the row D_{xy} are equal to $2 * (C - 0.5)$ where C is the C-index or concordance probability.

For validate.cph with dxy=TRUE, you must specify an argument u if the model is stratified, since survival curves can then cross and $X\beta$ is not 1-1 with predicted survival.
There is also validate method for tree, which only does cross-validation and which has a different list of arguments.

Value

a matrix with rows corresponding to the statistical indexes and columns for columns for the original index, resample estimates, indexes applied to the whole or omitted sample using the model derived from the resample, average optimism, corrected index, and number of successful re-samples.

Side Effects

prints a summary, and optionally statistics for each re-fit

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

validate.ols, validate.cph, validate.lrm, validate.rpart, predab.resample, fastbw, rms, rms.trans, calibrate, dxy.cens, concordancefit

Examples

```
# See examples for validate.cph, validate.lrm, validate.ols
# Example of validating a parametric survival model:

require(survival)
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('Male','Female'), n, TRUE))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
dt <- -log(runif(n))/h
e <- ifelse(dt <= cens,1,0)
dt <- pmin(dt, cens)
units(dt) <- "Year"
S <- Surv(dt,e)


f <- psm(S ~ age*sex, x=TRUE, y=TRUE)  # Weibull model
# Validate full model fit
validate(f, B=10)                # usually B=150


# Validate stepwise model with typical (not so good) stopping rule
# bw=TRUE does not preserve hierarchy of terms at present
validate(f, B=10, bw=TRUE, rule="p", sls=.1, type="individual")
```

---

validate.cph *Validation of a Fitted Cox or Parametric Survival Model's Indexes of Fit*

Description

This is the version of the `validate` function specific to models fitted with `cph` or `psm`. Also included is a small function `dxy.cens` that retrieves D_{xy} and its standard error from the `survival` package's `concordancefit` function. This allows for incredibly fast computation of D_{xy} or the c-index even for hundreds of thousands of observations. `dxy.cens` negates D_{xy} if log relative hazard is being predicted. If `y` is a left-censored `Surv` object, times are negated and a right-censored object is created, then D_{xy} is negated.

See `predab.resample` for information about confidence limits.

Usage

```
# fit <- cph(formula=Surv(ftime,event) ~ terms, x=TRUE, y=TRUE, \dots)
## S3 method for class 'cph'
validate(fit, method="boot", B=40, bw=FALSE, rule="aic",
type="residual", sls=.05, aics=0, force=NULL, estimates=TRUE,
pr=FALSE, dxy=TRUE, u, tol=1e-9, ...)

## S3 method for class 'psm'
validate(fit, method="boot",B=40,
        bw=FALSE, rule="aic", type="residual", sls=.05, aics=0,
        force=NULL, estimates=TRUE, pr=FALSE,
        dxy=TRUE, tol=1e-12, rel.tolerance=1e-5, maxiter=15, ...)

dxy.cens(x, y, type=c('time','hazard'))
```

Arguments

`fit`	a fit derived `cph`. The options `x=TRUE` and `y=TRUE` must have been specified. If the model contains any stratification factors and dxy=TRUE, the options `surv=TRUE` and `time.inc=u` must also have been given, where `u` is the same value of `u` given to `validate`.
`method`	see `validate`
`B`	number of repetitions. For `method="crossvalidation"`, is the number of groups of omitted observations.
`rel.tolerance`, `maxiter`, `bw`	`TRUE` to do fast step-down using the `fastbw` function, for both the overall model and for each repetition. `fastbw` keeps parameters together that represent the same factor.
`rule`	Applies if `bw=TRUE`. `"aic"` to use Akaike's information criterion as a stopping rule (i.e., a factor is deleted if the χ^2 falls below twice its degrees of freedom), or `"p"` to use P-values.
`type`	`"residual"` or `"individual"` - stopping rule is for individual factors or for the residual χ^2 for all variables deleted. For `dxy.cens`, specify `type="hazard"` if `x` is on the hazard or cumulative hazard (or their logs) scale, causing negation of the correlation index.
`sls`	significance level for a factor to be kept in a model, or for judging the residual χ^2.

aics	cutoff on AIC when rule="aic".
force	see fastbw
estimates	see print.fastbw
pr	TRUE to print results of each repetition
tol, ...	see validate or predab.resample
dxy	set to TRUE to validate Somers' D_{xy} using dxy.cens, which is fast until n > 500,000. Uses the survival package's concordancefit service function for concordance.
u	must be specified if the model has any stratification factors and dxy=TRUE. In that case, strata are not included in $X\beta$ and the survival curves may cross. Predictions at time t=u are correlated with observed survival times. Does not apply to validate.psm.
x	a numeric vector
y	a Surv object that may be uncensored or right-censored

Details

Statistics validated include the Nagelkerke R^2, D_{xy}, slope shrinkage, the discrimination index D [(model L.R. χ^2 - 1)/L], the unreliability index U = (difference in -2 log likelihood between uncalibrated $X\beta$ and $X\beta$ with overall slope calibrated to test sample) / L, and the overall quality index $Q = D - U$. g is the g-index on the log relative hazard (linear predictor) scale. L is -2 log likelihood with beta=0. The "corrected" slope can be thought of as shrinkage factor that takes into account overfitting. See predab.resample for the list of resampling methods.

Value

matrix with rows corresponding to D_{xy}, Slope, D, U, and Q, and columns for the original index, resample estimates, indexes applied to whole or omitted sample using model derived from resample, average optimism, corrected index, and number of successful resamples.

The values corresponding to the row D_{xy} are equal to $2 * (C - 0.5)$ where C is the C-index or concordance probability. If the user is correlating the linear predictor (predicted log hazard) with survival time, D_{xy} is automatically negated.

Side Effects

prints a summary, and optionally statistics for each re-fit (if pr=TRUE)

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

validate, predab.resample, fastbw, rms, rms.trans, calibrate, rcorr.cens, cph, survival-internal, gIndex, concordancefit

Examples

```
require(survival)
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('Male','Female'), n, TRUE))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
dt <- -log(runif(n))/h
e <- ifelse(dt <= cens,1,0)
dt <- pmin(dt, cens)
units(dt) <- "Year"
S <- Surv(dt,e)

f <- cph(S ~ age*sex, x=TRUE, y=TRUE)
# Validate full model fit
validate(f, B=10)               # normally B=150

# Validate a model with stratification.  Dxy is the only
# discrimination measure for such models, by Dxy requires
# one to choose a single time at which to predict S(t|X)
f <- cph(S ~ rcs(age)*strat(sex),
         x=TRUE, y=TRUE, surv=TRUE, time.inc=2)
validate(f, u=2, B=10)   # normally B=150
# Note u=time.inc
```

`validate.lrm` *Resampling Validation of a Logistic or Ordinal Regression Model*

Description

The `validate` function when used on an object created by `lrm` or `orm` does resampling validation of a logistic regression model, with or without backward step-down variable deletion. It provides bias-corrected Somers' D_{xy} rank correlation, R-squared index, the intercept and slope of an overall logistic calibration equation, the maximum absolute difference in predicted and calibrated probabilities E_{max}, the discrimination index D (model L.R. $(\chi^2 - 1)/n$), the unreliability index U = difference in -2 log likelihood between un-calibrated $X\beta$ and $X\beta$ with overall intercept and slope calibrated to test sample / n, the overall quality index (logarithmic probability score) $Q = D - U$, and the Brier or quadratic probability score, B (the last 3 are not computed for ordinal models), the g-index, and gp, the g-index on the probability scale. The corrected slope can be thought of as shrinkage factor that takes into account overfitting. For `orm` fits, a subset of the above indexes is provided, Spearman's ρ is substituted for D_{xy}, and a new index is reported: `pdm`, the mean absolute difference between 0.5 and the predicted probability that $Y \geq$ the marginal median of Y.

See `predab.resample` for information about confidence limits.

Usage

```
# fit <- lrm(formula=response ~ terms, x=TRUE, y=TRUE) or orm
## S3 method for class 'lrm'
validate(fit, method="boot", B=40,
         bw=FALSE, rule="aic", type="residual", sls=0.05, aics=0,
         force=NULL, estimates=TRUE,
         pr=FALSE,  kint, Dxy.method,
         emax.lim=c(0,1), ...)
## S3 method for class 'orm'
validate(fit, method="boot", B=40, bw=FALSE, rule="aic",
         type="residual",sls=.05, aics=0, force=NULL, estimates=TRUE,
         pr=FALSE,  ...)
```

Arguments

`fit`	a fit derived by `lrm` or `orm`. The options `x=TRUE` and `y=TRUE` must have been specified.
`method`, `B`, `bw`, `rule`, `type`, `sls`, `aics`, `force`, `estimates`, `pr`	see `validate` and `predab.resample`
`kint`	In the case of an ordinal model, specify which intercept to validate. Default is the middle intercept. For `validate.orm`, intercept-specific quantities are not validated so this does not matter.
`Dxy.method`	deprecated and ignored. `lrm` through `lrm.fit` computes exact rank correlation coefficients as of version 6.9-0.
`emax.lim`	range of predicted probabilities over which to compute the maximum error. Default is entire range.
`...`	other arguments to pass to `lrm.fit` and to `predab.resample` (note especially the `group`, `cluster`, and `subset` parameters)

Details

If the original fit was created using penalized maximum likelihood estimation, the same `penalty.matrix` used with the original fit are used during validation.

See https://www.fharrell.com/post/bootcal/ for simulations of the accuracy of approximate bootstrap confidence intervals for overfitting-corrected Brier scores.

Value

a matrix with rows corresponding to D_{xy}, R^2, `Intercept`, `Slope`, E_{max}, D, U, Q, B, g, gp, and columns for the original index, resample estimates, indexes applied to the whole or omitted sample using the model derived from the resample, average optimism, corrected index, and number of successful re-samples. For `validate.orm` not all columns are provided, Spearman's rho is returned instead of D_{xy}, and `pdm` is reported.

Side Effects

prints a summary, and optionally statistics for each re-fit

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

References

Miller ME, Hui SL, Tierney WM (1991): Validation techniques for logistic regression models. Stat in Med 10:1213–1226.

Harrell FE, Lee KL (1985): A comparison of the *discrimination* of discriminant analysis and logistic regression under multivariate normality. In Biostatistics: Statistics in Biomedical, Public Health, and Environmental Sciences. The Bernard G. Greenberg Volume, ed. PK Sen. New York: North-Holland, p. 333–343.

See Also

predab.resample, fastbw, lrm, rms, rms.trans, calibrate, somers2, cr.setup, gIndex, orm

Examples

```
n <- 1000    # define sample size
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))


# Specify population model for log odds that Y=1
L <- .4*(sex=='male') + .045*(age-50) +
  (log(cholesterol - 10)-5.2)*(-2*(sex=='female') + 2*(sex=='male'))
# Simulate binary y to have Prob(y=1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)


f <- lrm(y ~ sex*rcs(cholesterol)+pol(age,2)+blood.pressure, x=TRUE, y=TRUE)
#Validate full model fit
validate(f, B=10)              # normally B=300
validate(f, B=10, group=y)
# two-sample validation: make resamples have same numbers of
# successes and failures as original sample


#Validate stepwise model with typical (not so good) stopping rule
validate(f, B=10, bw=TRUE, rule="p", sls=.1, type="individual")


## Not run:
#Fit a continuation ratio model and validate it for the predicted
#probability that y=0
u <- cr.setup(y)
Y <- u$y
```

```
cohort <- u$cohort
attach(mydataframe[u$subs,])
f <- lrm(Y ~ cohort+rcs(age,4)*sex, penalty=list(interaction=2))
validate(f, cluster=u$subs, subset=cohort=='all')
#see predab.resample for cluster and subset

## End(Not run)
```

---

validate.ols *Validation of an Ordinary Linear Model*

Description

The validate function when used on an object created by ols does resampling validation of a multiple linear regression model, with or without backward step-down variable deletion. Uses resampling to estimate the optimism in various measures of predictive accuracy which include R^2, MSE (mean squared error with a denominator of n), the g-index, and the intercept and slope of an overall calibration $a + b\hat{y}$. The "corrected" slope can be thought of as shrinkage factor that takes into account overfitting. validate.ols can also be used when a model for a continuous response is going to be applied to a binary response. A Somers' D_{xy} for this case is computed for each resample by dichotomizing y. This can be used to obtain an ordinary receiver operating characteristic curve area using the formula $0.5(D_{xy} + 1)$. The Nagelkerke-Maddala R^2 index for the dichotomized y is also given. See predab.resample for information about confidence limits and for the list of resampling methods.

The LaTeX needspace package must be in effect to use the latex method.

Usage

```
# fit <- fitting.function(formula=response ~ terms, x=TRUE, y=TRUE)
## S3 method for class 'ols'
validate(fit, method="boot", B=40,
         bw=FALSE, rule="aic", type="residual", sls=0.05, aics=0,
         force=NULL, estimates=TRUE, pr=FALSE, u=NULL, rel=">",
         tolerance=1e-7, ...)
```

Arguments

fit	a fit derived by ols. The options x=TRUE and y=TRUE must have been specified. See validate for a description of arguments method - pr.
method, B, bw, rule, type, sls, aics, force, estimates, pr	see validate and predab.resample and fastbw
u	If specifed, y is also dichotomized at the cutoff u for the purpose of getting a bias-corrected estimate of D_{xy}.
rel	relationship for dichotomizing predicted y. Defaults to ">" to use y>u. rel can also be "<", ">=", and "<=".
tolerance	tolerance for singularity; passed to lm.fit.qr.
...	other arguments to pass to predab.resample, such as group, cluster, and subset

Value

matrix with rows corresponding to R-square, MSE, g, intercept, slope, and optionally D_{xy} and R^2, and columns for the original index, resample estimates, indexes applied to whole or omitted sample using model derived from resample, average optimism, corrected index, and number of successful resamples.

Side Effects

prints a summary, and optionally statistics for each re-fit

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

ols, predab.resample, fastbw, rms, rms.trans, calibrate, gIndex

Examples

```
set.seed(1)
x1 <- runif(200)
x2 <- sample(0:3, 200, TRUE)
x3 <- rnorm(200)
distance <- (x1 + x2/3 + rnorm(200))^2

f <- ols(sqrt(distance) ~ rcs(x1,4) + scored(x2) + x3, x=TRUE, y=TRUE)

#Validate full model fit (from all observations) but for x1 < .75
validate(f, B=20, subset=x1 < .75)   # normally B=300

#Validate stepwise model with typical (not so good) stopping rule
validate(f, B=20, bw=TRUE, rule="p", sls=.1, type="individual")
```

---

validate.rpart — *Dxy and Mean Squared Error by Cross-validating a Tree Sequence*

Description

Uses xval-fold cross-validation of a sequence of trees to derive estimates of the mean squared error and Somers' Dxy rank correlation between predicted and observed responses. In the case of a binary response variable, the mean squared error is the Brier accuracy score. For survival trees, Dxy is negated so that larger is better. There are print and plot methods for objects created by validate.rpart.

Usage

```
# f <- rpart(formula=y ~ x1 + x2 + \dots) # or rpart
## S3 method for class 'rpart'
validate(fit, method, B, bw, rule, type, sls, aics,
    force, estimates, pr=TRUE,
    k, rand, xval=10, FUN, ...)
## S3 method for class 'validate.rpart'
print(x, ...)
## S3 method for class 'validate.rpart'
plot(x, what=c("mse","dxy"), legendloc=locator, ...)
```

Arguments

`fit`	an object created by `rpart`. You must have specified the `model=TRUE` argument to `rpart`.
`method`, `B`, `bw`, `rule`, `type`, `sls`, `aics`, `force`, `estimates`	are there only for consistency with the generic `validate` function; these are ignored
`x`	the result of `validate.rpart`
`k`	a sequence of cost/complexity values. By default these are obtained from calling `FUN` with no optional arguments or from the `rpart` `cptable` object in the original fit object. You may also specify a scalar or vector.
`rand`	a random sample (usually omitted)
`xval`	number of splits
`FUN`	the name of a function which produces a sequence of trees, such `prune`.
`...`	additional arguments to `FUN` (ignored by `print,plot`).
`pr`	set to `FALSE` to prevent intermediate results for each `k` to be printed
`what`	a vector of things to plot. By default, 2 plots will be done, one for `mse` and one for `Dxy`.
`legendloc`	a function that is evaluated with a single argument equal to `1` to generate a list with components `x, y` specifying coordinates of the upper left corner of a legend, or a 2-vector. For the latter, `legendloc` specifies the relative fraction of the plot at which to center the legend.

Value

a list of class `"validate.rpart"` with components named `k, size, dxy.app, dxy.val, mse.app, mse.val, binary, xval`. `size` is the number of nodes, `dxy` refers to Somers' `D`, `mse` refers to mean squared error of prediction, `app` means apparent accuracy on training samples, `val` means validated accuracy on test samples, `binary` is a logical variable indicating whether or not the response variable was binary (a logical or 0/1 variable is binary). `size` will not be present if the user specifies `k`.

Side Effects

prints if `pr=TRUE`

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

See Also

rpart, somers2, dxy.cens, locator, legend

Examples

```
## Not run:
n <- 100
set.seed(1)
x1 <- runif(n)
x2 <- runif(n)
x3 <- runif(n)
y  <- 1*(x1+x2+rnorm(n) > 1)
table(y)
require(rpart)
f <- rpart(y ~ x1 + x2 + x3, model=TRUE)
v <- validate(f)
v    # note the poor validation
par(mfrow=c(1,2))
plot(v, legendloc=c(.2,.5))
par(mfrow=c(1,1))

## End(Not run)
```

---

validate.Rq *Validation of a Quantile Regression Model*

Description

The validate function when used on an object created by Rq does resampling validation of a quantile regression model, with or without backward step-down variable deletion. Uses resampling to estimate the optimism in various measures of predictive accuracy which include mean absolute prediction error (MAD), Spearman rho, the g-index, and the intercept and slope of an overall calibration $a + b\hat{y}$. The "corrected" slope can be thought of as shrinkage factor that takes into account overfitting. validate.Rq can also be used when a model for a continuous response is going to be applied to a binary response. A Somers' D_{xy} for this case is computed for each resample by dichotomizing y. This can be used to obtain an ordinary receiver operating characteristic curve area using the formula $0.5(D_{xy} + 1)$. See predab.resample for information about confidence limits and for the list of resampling methods.

The LaTeX needspace package must be in effect to use the latex method.

Usage

```
# fit <- fitting.function(formula=response ~ terms, x=TRUE, y=TRUE)
## S3 method for class 'Rq'
validate(fit, method="boot", B=40,
         bw=FALSE, rule="aic", type="residual", sls=0.05, aics=0,
         force=NULL, estimates=TRUE, pr=FALSE, u=NULL, rel=">",
         tolerance=1e-7, ...)
```

Arguments

`fit`	a fit derived by `Rq`. The options `x=TRUE` and `y=TRUE` must have been specified. See `validate` for a description of arguments `method` - `pr`.
`method`, `B`, `bw`, `rule`, `type`, `sls`, `aics`, `force`, `estimates`, `pr`	see `validate` and `predab.resample` and `fastbw`
`u`	If specifed, `y` is also dichotomized at the cutoff `u` for the purpose of getting a bias-corrected estimate of D_{xy}.
`rel`	relationship for dichotomizing predicted `y`. Defaults to `">"` to use `y>u`. `rel` can also be `"<"`, `">="`, and `"<="`.
`tolerance`	ignored
`...`	other arguments to pass to `predab.resample`, such as `group`, `cluster`, and `subset`

Value

matrix with rows corresponding to various indexes, and optionally D_{xy}, and columns for the original index, resample estimates, indexes applied to whole or omitted sample using model derived from resample, average optimism, corrected index, and number of successful resamples.

Side Effects

prints a summary, and optionally statistics for each re-fit

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

See Also

`Rq`, `predab.resample`, `fastbw`, `rms`, `rms.trans`, `gIndex`

Examples

```
set.seed(1)
x1 <- runif(200)
x2 <- sample(0:3, 200, TRUE)
x3 <- rnorm(200)
```

```
distance <- (x1 + x2/3 + rnorm(200))^2

f <- Rq(sqrt(distance) ~ rcs(x1,4) + scored(x2) + x3, x=TRUE, y=TRUE)

#Validate full model fit (from all observations) but for x1 < .75
validate(f, B=20, subset=x1 < .75)   # normally B=300

#Validate stepwise model with typical (not so good) stopping rule
validate(f, B=20, bw=TRUE, rule="p", sls=.1, type="individual")
```

`vif` *Variance Inflation Factors*

Description

Computes variance inflation factors from the covariance matrix of parameter estimates, using the method of Davis et al. (1986), which is based on the correlation matrix from the information matrix.

Usage

```
vif(fit)
```

Arguments

`fit`	an object created by `lrm`, `ols`, `psm`, `cph`, `Rq`, `Glm`, `glm`

Value

vector of vifs

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
fh@fharrell.com

References

Davis CE, Hyde JE, Bangdiwala SI, Nelson JJ: An example of dependencies among variables in a conditional logistic regression. In Modern Statistical Methods in Chronic Disease Epidemiology, Eds SH Moolgavkar and RL Prentice, pp. 140–147. New York: Wiley; 1986.

See Also

`rmsMisc` (for `num.intercepts`

Examples

```
set.seed(1)
x1 <- rnorm(100)
x2 <- x1+.1*rnorm(100)
y  <- sample(0:1, 100, TRUE)
f  <- lrm(y ~ x1 + x2)
vif(f)
```

`which.influence` *Which Observations are Influential*

Description

Creates a list with a component for each factor in the model. The names of the components are the factor names. Each component contains the observation identifiers of all observations that are "overly influential" with respect to that factor, meaning that $|dfbetas| > u$ for at least one β_i associated with that factor, for a given `cutoff`. The default `cutoff` is `.2`. The fit must come from a function that has `resid(fit, type="dfbetas")` defined.

`show.influence`, written by Jens Oehlschlaegel-Akiyoshi, applies the result of `which.influence` to a data frame, usually the one used to fit the model, to report the results.

Usage

```
which.influence(fit, cutoff=.2)

show.influence(object, dframe, report=NULL, sig=NULL, id=NULL)
```

Arguments

`fit`	fit object
`object`	the result of `which.influence`
`dframe`	data frame containing observations pertinent to the model fit
`cutoff`	cutoff value
`report`	other columns of the data frame to report besides those corresponding to predictors that are influential for some observations
`sig`	runs results through `signif` with `sig` digits if `sig` is given
`id`	a character vector that labels rows of `dframe` if `row.names` were not used

Value

`show.influence` returns a marked dataframe with the first column being a count of influence values

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
fh@fharrell.com

Jens Oehlschlaegel-Akiyoshi
Center for Psychotherapy Research
Christian-Belser-Strasse 79a
D-70597 Stuttgart Germany
oehl@psyres-stuttgart.de

See Also

residuals.lrm, residuals.cph, residuals.ols, rms, lrm, ols, cph

Examples

```
#print observations in data frame that are influential,
#separately for each factor in the model
x1 <- 1:20
x2 <- abs(x1-10)
x3 <- factor(rep(0:2,length.out=20))
y  <- c(rep(0:1,8),1,1,1,1)
f  <- lrm(y ~ rcs(x1,3) + x2 + x3, x=TRUE,y=TRUE)
w <- which.influence(f, .55)
nam <- names(w)
d   <- data.frame(x1,x2,x3,y)
for(i in 1:length(nam)) {
 print(paste("Influential observations for effect of ",nam[i]),quote=FALSE)
 print(d[w[[i]],])
}

show.influence(w, d)  # better way to show results
```

Xcontrast *Xcontrast*

Description

Produce Design Matrices for Contrasts

Usage

```
Xcontrast(
  fit,
  a,
  b = NULL,
  a2 = NULL,
```

```
  b2 = NULL,
  ycut = NULL,
  weights = "equal",
  expand = TRUE,
  Zmatrix = TRUE
)
```

Arguments

`fit`	an ‘rms‘ or ‘rmsb‘ fit object, not necessarily complete
`a`	see [rms::contrast.rms()]
`b`	see [rms::contrast.rms()]
`a2`	see [rms::contrast.rms()]
`b2`	see [rms::contrast.rms()]
`ycut`	see [rms::contrast.rms()]
`weights`	see [rms::contrast.rms()]
`expand`	see [rms::contrast.rms()]
`Zmatrix`	set to ‘FALSE‘ for a partial PO model in which you do not want to include the Z matrix in the returned contrast matrix

Details

This is a simpler version of ‘contrast.rms‘ that creates design matrices or differences of them and does not require the fit object to be complete (i.e., to have coefficients). This is used for the ‘pcontrast‘ option in [rmsb::blrm()].

Value

numeric matrix

Author(s)

Frank Harrell

`[.Ocens` *Ocens*

Description

Subset Method for `Ocens` Objects

Usage

```
## S3 method for class 'Ocens'
x[..., drop]
```

Arguments

`x`	an `Ocens` object
`...`	the usual rows and columns specifiers
`drop`	set to `FALSE` to not drop unneeded dimensions

Details

Subsets an `Ocens` object, preserving its special attributes. Attributes are not updated. In the future such updating should be implemented.

Value

new `Ocens` object or by default an unclassed vector if only one column of `x` is being kept

Author(s)

Frank Harrell

Index

∗ **aggregation**
groupkm, 81
∗ **aplot**
anova.rms, 5
∗ **array**
matinv, 112
∗ **bootstrap**
anova.rms, 5
bootBCa, 17
bootcov, 18
calibrate, 30
predab.resample, 167
validate, 278
validate.cph, 280
validate.lrm, 283
validate.ols, 286
validate.Rq, 289
∗ **calibration**
calibrate, 30
∗ **category**
cr.setup, 50
impactPO, 86
lrm, 98
orm, 133
plot.xmean.ordinaly, 159
validate.rpart, 287
∗ **character**
latex.cph, 94
latexrms, 96
∗ **cluster sampling**
robcov, 236
∗ **continuation ratio model**
cr.setup, 50
lrm, 98
∗ **grouping**
groupkm, 81
∗ **hplot**
bootcov, 18
bplot, 27
calibrate, 30
ggplot.Predict, 65
nomogram, 114
plot.Predict, 149
plot.xmean.ordinaly, 159
plotp.Predict, 162
summary.rms, 248
survplot, 259
∗ **htest**
anova.rms, 5
bootcov, 18
bplot, 27
contrast.rms, 34
fastbw, 57
ggplot.Predict, 65
plot.Predict, 149
plotp.Predict, 162
sensuc, 241
summary.rms, 248
val.prob, 269
∗ **interface**
Function, 59
latex.cph, 94
latexrms, 96
summary.rms, 248
∗ **intra-class correlation**
robcov, 236
∗ **logistic regression model**
cr.setup, 50
Function, 59
lrm, 98
orm, 133
orm.fit, 141
pentrace, 144
plot.xmean.ordinaly, 159
predict.lrm, 177
residuals.lrm, 205
rms, 213
rms.trans, 216

sensuc, 241
summary.rms, 248
val.prob, 269
validate.lrm, 283
which.influence, 292
∗ **logistic**
lrm.fit, 107
∗ **manip**
gendata, 61
rms, 213
rms.trans, 216
∗ **math**
Function, 59
rms, 213
rms.trans, 216
∗ **methods**
bootcov, 18
calibrate, 30
Function, 59
gendata, 61
latexrms, 96
rms, 213
rms.trans, 216
rmsMisc, 219
specs.rms, 247
validate, 278
∗ **model reliability**
calibrate, 30
∗ **model validation**
calibrate, 30
plot.xmean.ordinaly, 159
predab.resample, 167
residuals.cph, 202
residuals.lrm, 205
residuals.ols, 210
sensuc, 241
val.prob, 269
val.surv, 273
validate, 278
validate.cph, 280
validate.lrm, 283
validate.ols, 286
validate.rpart, 287
validate.Rq, 289
∗ **models**
anova.rms, 5
bj, 13
bootcov, 18
bplot, 27
calibrate, 30
contrast.rms, 34
cph, 43
cr.setup, 50
datadist, 52
fastbw, 57
Function, 59
gendata, 61
ggplot.Predict, 65
Glm, 77
Gls, 79
impactPO, 86
latex.cph, 94
latexrms, 96
lrm, 98
lrm.fit, 107
nomogram, 114
ols, 127
orm, 133
orm.fit, 141
pentrace, 144
plot.Predict, 149
plot.xmean.ordinaly, 159
plotp.Predict, 162
pphsm, 166
predab.resample, 167
Predict, 171
predict.lrm, 177
predictrms, 180
psm, 195
residuals.lrm, 205
residuals.ols, 210
rms, 213
rms.trans, 216
rmsMisc, 219
rmsOverview, 226
robcov, 236
Rq, 239
sensuc, 241
specs.rms, 247
summary.rms, 248
survest.cph, 252
survest.psm, 256
survplot, 259
val.prob, 269
val.surv, 273
validate, 278

validate.cph, 280
validate.lrm, 283
validate.ols, 286
validate.rpart, 287
validate.Rq, 289
vif, 291
which.influence, 292
∗ **nonparametric**
cph, 43
datadist, 52
groupkm, 81
Rq, 239
survplot, 259
∗ **ordinal logistic model**
cr.setup, 50
lrm, 98
orm, 133
∗ **ordinal response**
cr.setup, 50
lrm, 98
orm, 133
∗ **overview**
rmsOverview, 226
∗ **penalized MLE**
pentrace, 144
∗ **predictive accuracy**
calibrate, 30
gIndex, 73
predab.resample, 167
val.prob, 269
val.surv, 273
validate, 278
validate.cph, 280
validate.lrm, 283
validate.ols, 286
validate.rpart, 287
validate.Rq, 289
∗ **print**
print.cph, 187
print.ols, 190
∗ **proportional odds model**
lrm, 98
orm, 133
∗ **regression**
anova.rms, 5
bootcov, 18
calibrate, 30
contrast.rms, 34
cr.setup, 50
datadist, 52
fastbw, 57
Function, 59
gendata, 61
Glm, 77
impactPO, 86
latex.cph, 94
latexrms, 96
lrm.fit, 107
nomogram, 114
ols, 127
orm.fit, 141
pentrace, 144
plot.xmean.ordinaly, 159
pphsm, 166
predict.lrm, 177
predictrms, 180
residuals.lrm, 205
residuals.ols, 210
rms, 213
rms.trans, 216
robcov, 236
sensuc, 241
specs.rms, 247
summary.rms, 248
survest.cph, 252
survest.psm, 256
val.prob, 269
val.surv, 273
validate, 278
validate.cph, 280
validate.lrm, 283
validate.ols, 286
validate.Rq, 289
vif, 291
which.influence, 292
∗ **ridge regression**
pentrace, 144
∗ **robust**
gIndex, 73
robcov, 236
∗ **sampling**
bootcov, 18
sensuc, 241
val.prob, 269
∗ **sensitivity analysis**
sensuc, 241

∗ **shrinkage**
pentrace, 144
∗ **smooth**
rms.trans, 216
val.prob, 269
val.surv, 273
∗ **stepwise**
fastbw, 57
∗ **stratification**
groupkm, 81
∗ **survival**
bj, 13
calibrate, 30
cph, 43
Function, 59
groupkm, 81
hazard.ratio.plot, 83
ie.setup, 85
latex.cph, 94
pphsm, 166
psm, 195
residuals.cph, 202
rms, 213
rms.trans, 216
sensuc, 241
summary.rms, 248
survest.cph, 252
survest.psm, 256
survfit.cph, 258
survplot, 259
val.surv, 273
validate, 278
validate.cph, 280
which.influence, 292
∗ **transformation**
rms.trans, 216
∗ **tree**
validate.rpart, 287
∗ **univar**
gIndex, 73
∗ **utilities**
setPb, 245
∗ **variable selection**
fastbw, 57
[.Ocens, 294
%ia% *(rms.trans)*, 216

abbreviate, *116*
abs.error.pred, *129*
adapt_orm, 4
AIC.rms *(rmsMisc)*, 219
anova.lm, *9*
anova.rms, 5, *38*, *47*, *67*, *69*, *129*, *151*, *153*, *215*, *225*
approx, *119*
as.data.frame.Ocens, 12
asis *(rms.trans)*, 216
axis, *119*

biVar, *160*
bj, 13
bjplot *(bj)*, 13
boot.ci, *18*, *175*
bootBCa, 17, *175*
bootcov, 18, *38*, *171*, *175*, *225*, *238*
bootcov(), *212*
bootplot *(bootcov)*, 18
boxplot, *207*
bplot, 27

calibrate, 30, *47*, *102*, *129*, *138*, *215*, *276*, *280*, *282*, *285*, *287*
calibrate.rms *(rmsMisc)*, 219
catg *(rms.trans)*, 216
combineRelatedPredictors *(rmsMisc)*, 219
concordance, *47*
concordancefit, *280*, *282*
confplot *(bootcov)*, 18
contourplot, *29*
contrast *(contrast.rms)*, 34
contrast.rms, *9*, *22*, 34, *69*, *153*, *163*, *175*, *179*, *183*, *251*
corClasses, *81*
cox.zph, *47*, *84*, *203*
coxph, *47*, *84*, *86*, *122*, *188*, *203*
cph, *16*, *33*, 43, *59*, *84*, *86*, *129*, *171*, *203*, *215*, *218*, *244*, *254*, *264*, *276*, *282*, *293*
cr.setup, 50, *86*, *102*, *160*, *285*
cr.setup(), *111*
cut2, *82*

datadist, *16*, *29*, *47*, 52, *69*, *114*, *118*, *129*, *153*, *163*, *171*, *175*, *183*, *198*, *215*, *218*, *225*, *247*, *251*, *264*
dataRep, *215*
de, *63*
describe, *54*, *215*
Design *(rms)*, 213

DesignAssign (rmsMisc), 219
dotchart2, *9*, *74*
dxy.cens, *16*, *47*, *280*, *289*
dxy.cens (validate.cph), 280

effective.df (pentrace), 144
errbar, *33*, *82*, *264*
expand.grid, *63*
ExProb, 55

factor(), *201*
fastbw, *47*, 57, *129*, *169*, *171*, *215*, *225*, *279*, *280*, *282*, *285–287*, *290*
fit.mult.impute, *33*
formatNP (rmsMisc), 219
Function, 59
Function.rms, *183*

gendata, *22*, *38*, 61, *183*, *215*
Getlim (rmsMisc), 219
Getlimi (rmsMisc), 219
ggplot, *69*, *89*, *90*
ggplot (importedexported), 89
ggplot.npsurv, 63, *122*
ggplot.Predict, *9*, *47*, 65, *119*, *153*, *163*, *175*, *183*, *198*, *264*
ggplot2::facet_wrap(), *149*
gIndex, 73, *102*, *138*, *144*, *183*, *282*, *285*, *287*, *290*
gIndex(), *111*
GiniMd, *16*, *47*, *75*, *81*, *198*
Glm, 77
glm, *51*, *102*, *144*
Gls, 79
gls, *81*
glsControl, *81*
glsObject, *81*
groupkm, *33*, 81, *276*
gTrans (rms.trans), 216

hare, *32*, *275*, *276*
Hazard (psm), 195
hazard.ratio.plot, 83
histdensity (bootcov), 18
histSpikeg, *68*, *69*, *163*
Hmisc::combine.levels(), *87*, *88*
Hmisc::cutGn(), *213*
Hmisc::fit.mult.impute(), *193*, *195*
Hmisc::GiniMd(), *78*
Hmisc::movStats(), *92*, *93*
Hmisc::propsPO(), *88*
Hmisc::qrxcenter(), *111*
Hmisc::R2Measures(), *88*, *112*
Hmisc::units(), *200*
html.anova.rms (anova.rms), 5
html.naprint.delete (rmsMisc), 219
html.summary.rms (summary.rms), 248
html.validate (validate), 278

ie.setup, *47*, 85
image, *147*
impactPO, 86
importedexported, 89
infoMxop, 90
infoMxop(), *110*, *111*
intCalibration, 92
interactions.containing (rmsMisc), 219
is.na.Ocens, 94

labcurve, *69*, *153*, *207*, *264*, *272*
latex, *8*, *9*, *96*, *98*, *215*, *225*, *250*
latex.anova.rms (anova.rms), 5
latex.bj (latexrms), 96
latex.cph, *47*, 94
latex.default, *95*, *279*
latex.Glm (latexrms), 96
latex.Gls (latexrms), 96
latex.lrm, *102*
latex.lrm (latex.cph), 94
latex.naprint.delete (rmsMisc), 219
latex.ols, *129*
latex.ols (latex.cph), 94
latex.orm, *138*
latex.orm (latex.cph), 94
latex.pphsm (latex.cph), 94
latex.psm, *198*
latex.psm (latex.cph), 94
latex.Rq (Rq), 239
latex.summary.rms (summary.rms), 248
latex.validate (validate), 278
latexrms, *61*, *95*, *96*, 96, *119*, *215*, *240*, *247*
latexSN, *225*
latexTabular, *225*
legend, *289*
legend.nomabbrev (nomogram), 114
levelplot, *29*
lines, *122*

lines.residuals.psm.censored.normalized (psm), 195
lm, *129*, *191*
lm.fit, *22*, *129*
lm.influence, *211*
lm.wfit, *129*
locator, *9*, *289*
loess, *207*
logLik.Gls, *81*
logLik.Gls (rmsMisc), 219
logLik.ols (rmsMisc), 219
logLik.rms (rmsMisc), 219
lowess, *33*, *207*
lrm, *47*, *51*, *59*, 98, *113*, *129*, *135–137*, *144*, *147*, *160*, *171*, *179*, *207*, *215*, *218*, *244*, *272*, *285*, *293*
lrm(), *88*, *109*, *111*
lrm.fit, *22*, *98*, *102*, 107, *272*
lrm.fit(), *201*
lrtest, *9*
lrtest (rmsMisc), 219
LRupdate, 111
lsp (rms.trans), 216

makepredictcall, *218*
makepredictcall.rms (rms.trans), 216
matinv, 112
matrx (rms.trans), 216
Mean.cph (cph), 43
Mean.lrm, *102*
Mean.lrm (predict.lrm), 177
Mean.orm, *138*
Mean.orm (predict.lrm), 177
Mean.psm (psm), 195
mgp.axis, *264*
mgp.axis.labels, *264*
Misc, *251*
model.frame.default, *215*
modelData (rms), 213
movStats, *33*

na.delete, *16*, *47*, *80*, *102*, *129*, *138*, *198*
na.detail.response, *16*, *47*, *102*, *129*, *138*, *198*
naresid, *179*, *203*, *207*, *238*
Newlabels (rmsMisc), 219
Newlevels (rmsMisc), 219
nnet::multinom(), *88*
nobs.rms (rmsMisc), 219
nomogram, *55*, 114, *215*
npsurv, 121
num.intercepts, *138*, *291*

Ocens, 123, *275*
Ocens2ord, 123
Ocens2Surv, 126
Olinks, 126
ols, *59*, 127, *147*, *171*, *191*, *211*, *215*, *218*, *287*, *293*
oos.loglik (rmsMisc), 219
ordESS, 130, *138*
ordGroupBoot, *22*
ordParallel, 131
orm, *33*, *57*, 133, *144*, *147*, *179*, *207*, *241*, *285*
orm.fit, *22*, *134*, *138*, 141
orm.fit(), *201*
Overview, *69*, *153*, *163*

page, *63*
pantext, *9*
pantext (plot.Predict), 149
par, *117*, *264*
param.order (rmsMisc), 219
Penalty.matrix (rmsMisc), 219
Penalty.setup (rmsMisc), 219
pentrace, *102*, *129*, *138*, 144
perimeter (bplot), 27
perlcode (Function), 59
plot, *122*
plot.anova.rms (anova.rms), 5
plot.calibrate (calibrate), 30
plot.contrast.rms, 148
plot.ExProb (ExProb), 55
plot.gIndex (gIndex), 73
plot.lrm.partial (residuals.lrm), 205
plot.nomogram (nomogram), 114
plot.pentrace (pentrace), 144
plot.Predict, *9*, *47*, *69*, *119*, 149, *163*, *175*, *183*, *198*, *264*
plot.rexVar, 157
plot.sensuc (sensuc), 241
plot.summary.rms, *119*
plot.summary.rms (summary.rms), 248
plot.val.prob (val.prob), 269
plot.val.surv (val.surv), 273
plot.val.survh (val.surv), 273
plot.validate.rpart (validate.rpart), 287

plot.xmean.ordinaly, 159
plotIntercepts, 161
plotp.Predict, 162, *175*
pol (rms.trans), 216
polr, *138*
polspline::hare(), *92*
poma, 165
pphsm, 166, *198*
predab.resample, *15*, *22*, *30*, *33*, *51*, *86*, *102*, *138*, 167, *278*, *280–287*, *289*, *290*
Predict, *22*, *29*, *38*, *47*, *54*, *63*, *69*, *129*, *153*, *163*, 171, *215*
predict.bj (predictrms), 180
predict.cph (predictrms), 180
predict.Glm (predictrms), 180
predict.Gls (predictrms), 180
predict.lrm, *102*, 177, *183*
predict.ols (predictrms), 180
predict.orm, *138*, *183*
predict.orm (predict.lrm), 177
predict.psm (predictrms), 180
predict.rms, *61*, *63*, *75*, *129*, *179*
predict.rms (predictrms), 180
predict.Rq (Rq), 239
predictrms, *69*, *153*, *163*, *172*, *175*, 180, *215*, *254*, *258*, *264*
pretty, *8*, *116*, *119*, *251*
pretty(), *64*
print, *122*
print.anova.rms (anova.rms), 5
print.bj (bj), 13
print.calibrate (calibrate), 30
print.char.matrix, *225*
print.contrast.rms (contrast.rms), 34
print.cph, *47*, 187
print.datadist, *63*
print.datadist (datadist), 52
print.fastbw, *169*, *279*, *282*
print.fastbw (fastbw), 57
print.gIndex (gIndex), 73
print.Glm, 188
print.Gls (Gls), 79
print.impactPO, 189
print.lrm (lrm), 98
print.lrtest (rmsMisc), 219
print.nomogram (nomogram), 114
print.Ocens, 189
print.ols, *129*, 190
print.orm (orm), 133
print.pentrace (pentrace), 144
print.pphsm, *167*
print.pphsm (pphsm), 166
print.Predict (Predict), 171
print.psm (psm), 195
print.rexVar, 191
print.rms (rmsMisc), 219
print.Rq (Rq), 239
print.specs.rms (specs.rms), 247
print.summary.rms (summary.rms), 248
print.survest.psm (survest.psm), 256
print.val.prob (val.prob), 269
print.val.survh (val.surv), 273
print.validate (validate), 278
print.validate.rpart (validate.rpart), 287
prmiInfo, *9*, 192
prModFit, *16*, *81*, *100*, *102*, *135*, *138*, *188*, *191*, *198*, *241*
prModFit (rmsMisc), 219
prModFit(), *78*
processMI, *33*, 193
processMI.fit.mult.impute, 193
processMI.fit.mult.impute(), *112*, *193*
prStats (rmsMisc), 219
psm, *16*, *33*, *59*, *167*, 195, *258*, *264*, *276*
Punits, 200

Quantile.cph (cph), 43
Quantile.orm, *57*
Quantile.orm (orm), 133
Quantile.psm (psm), 195

R2Measures, *100*, *135*, *136*, *188*, *195*, *197*, *198*
rbind.Predict, *69*, *153*, *163*
rbind.Predict (Predict), 171
rcorr.cens, *16*, *282*
rcs (rms.trans), 216
rcspline.eval, *218*
rcspline.restate, *96*, *98*, *218*
recode2integer, *144*, 201
recode2integer(), *111*
related.predictors (rmsMisc), 219
reListclean (rmsMisc), 219
removeFormulaTerms (rmsMisc), 219
reShape, *183*
residuals.bj (bj), 13
residuals.coxph, *202*, *203*

residuals.cph, *47*, *84*, *183*, 202, *238*, *293*
residuals.Glm, 204
residuals.lrm, *102*, *160*, 205, *293*
residuals.ols, *129*, 210, *293*
residuals.orm, *138*
residuals.orm *(residuals.lrm)*, 205
residuals.psm *(psm)*, 195
residuals.survreg, *198*
rexVar, 212
rms, *9*, *16*, *22*, *29*, *47*, *54*, *59*, *61*, *69*, *98*, *102*, *119*, *129*, *138*, *153*, *163*, *171*, *175*, *183*, *198*, 213, *218*, *225*, *247*, *251*, *258*, *264*, *280*, *282*, *285*, *287*, *290*, *293*
rms.Overview *(rmsOverview)*, 226
rms.trans, *9*, *47*, *54*, *61*, *102*, *129*, *138*, *175*, *183*, *215*, 216, *247*, *251*, *280*, *282*, *285*, *287*, *290*
rmsArgs *(rmsMisc)*, 219
rmsb::pdensityContour(), *149*
rmsMisc, *9*, *22*, *29*, *59*, *63*, *69*, *102*, *119*, *138*, *147*, *153*, *163*, *175*, *215*, 219, *251*, *291*
rmsOverview, 226
robcov, *22*, 236
robcov(), *132*
round(), *191*
rpart, *289*
Rq, 239, *290*
rq, *240, 241*
RqFit *(Rq)*, 239

sample, *22*, *244*
sascode *(Function)*, 59
scales::breaks_pretty(), *64*
scat1d, *33*, *152, 153*, *272*, *276*
scored *(rms.trans)*, 216
sensuc, 241
setPb, *22*, *171*, 245
setTkProgressBar, *246*
show.influence *(which.influence)*, 292
solve, *113*, *138*, *144*
solvet, *9*, *59*, *147*
somers2, *285*, *289*
specs *(specs.rms)*, 247
specs.rms, *47*, *129*, *215*, 247
stats::glm(), *78*, *111*
stats::glm.fit(), *109*, *111*
stats::nlm(), *108*, *111*
stats::nlminb(), *108*, *111*
stats::optim(), *108*, *111*
stats::residuals.glm, *78*
strat *(rms.trans)*, 216
strata, *122*
summary.formula, *160*
summary.glm, *225*
summary.lm, *129*, *225*
summary.rms, *9*, *38*, *47*, *54*, *69*, *129*, *153*, *163*, *167*, *175*, *183*, *215*, 248
supsmu, *207*
Surv, *16*, *47*, *82*, *84*, *86*, *89, 90*, *198*, *275*
Surv *(importedexported)*, 89
survdiffplot *(survplot)*, 259
survest, *175*, *198*, *264*
survest *(survest.cph)*, 252
survest(), *92*
survest.cph, *47*, *63*, 252, *259*
survest.orm, 255
survest.psm, *63*, 256
survest.rms *(rmsMisc)*, 219
survfit, *82*, *258, 259*, *264*
survfit.coxph, *47*, *254*, *259*
survfit.cph, *47*, *122*, *254*, 258
survfit.formula, *121*
Survival *(psm)*, 195
Survival.cph *(cph)*, 43
Survival.orm *(ExProb)*, 55
survplot, *47*, *122*, *175*, *198*, *254*, *258*, 259
survplot.orm, 266
survplot.residuals.psm.censored.normalized *(psm)*, 195
survplotp *(survplot)*, 259
survreg, *16*, *198*, *258*
survreg.control, *32*
survreg.distributions, *198*, *258*, *264*
survreg.object, *198*

tkProgressBar, *246*
transace, *175*
transcan, *61*

unique(), *201*
units, *82*, *264*
univarLR *(rmsMisc)*, 219

val.prob, 269
val.surv, *33*, 273

validate, *31*, *33*, *47*, *59*, *129*, *171*, *215*, *276*, 278, *281*, *282*, *284*, *286*, *290*
validate.bj *(*bj*)*, 13
validate.cph, *280*, 280
validate.lrm, *102*, *272*, *280*, 283
validate.ols, *280*, 286
validate.orm, *138*
validate.orm *(*validate.lrm*)*, 283
validate.psm *(*validate.cph*)*, 280
validate.rpart, *280*, 287
validate.Rq, 289
varClasses, *81*
varFunc, *81*
vcov.cph *(*rmsMisc*)*, 219
vcov.Glm *(*rmsMisc*)*, 219
vcov.Gls *(*rmsMisc*)*, 219
vcov.lrm *(*rmsMisc*)*, 219
vcov.ols *(*rmsMisc*)*, 219
vcov.orm, *138*
vcov.orm *(*rmsMisc*)*, 219
vcov.pphsm *(*pphsm*)*, 166
vcov.psm *(*rmsMisc*)*, 219
vcov.rms *(*rmsMisc*)*, 219
VGAM::vglm(), *88*
vif, *47*, *102*, *129*, *138*, *215*, *225*, 291

which.influence, *47*, *129*, *207*, *211*, *215*, 292
wireframe, *28*, *29*
wtd.stats, *272*

Xcontrast, 293
xYplot, *9*, *153*, *183*
xyplot, *152*