The Design Package

June 18, 2007

Version 2.1-1

Date 2007-06-14

Title Design Package

Author Frank E Harrell Jr <f.harrell@vanderbilt.edu>

Maintainer Charles Dupont <charles.dupont@vanderbilt.edu>

Depends Hmisc (>= 3.4-0), survival

Suggests MASS, rpart, nlme

Description Regression modeling, testing, estimation, validation, graphics, prediction, and typesetting by storing enhanced model design

License GPL version 2 or newer

URL http://biostat.mc.vanderbilt.edu/s/Design, http://biostat.mc.vanderbilt.edu/rms

LazyLoad yes

R topics documented:

`Design-internal` *Internal Design functions*

Description

Internal Design functions.

Details

These are not to be called by the user or are undocumented.

`Design.Misc` *Miscellaneous Design Attributes and Utility Functions*

Description

These functions are used internally to `anova.Design`, `fastbw`, etc., to retrieve various attributes of a design. These functions allow some fitting functions not in the `Design` series (e.g,, `lm`, `glm`) to be used with `anova.Design`, `fastbw`, and similar functions.

For `Varcov`, there are these functions: `Varcov.default`, `Varcov.lm`, `Varcov.glm`. 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.

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).

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.

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).

`univarLR` takes a multivariable model fit object from `Design` 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.

`DesignFit` is used to convert a fit from non-Design functions (e.g., `glm`) that were invoked with Design in effect to Design functions so that `anova.Design` will be called by `anova()`, etc. So that the original fit's `residuals` and `print` methods, if they exist, will be called, there are functions `print.Design` and `residuals.Design` to dispatch them. These two functions are not needed in versions of S-Plus prior to 5.x (i.e., non-SV4).

Usage

```
## S3 method for class 'cph':
Varcov(object, regcoef.only=FALSE, ...)
## S3 method for class 'glmD':
Varcov(object, regcoef.only=FALSE, ...)
## S3 method for class 'glsD':
Varcov(object, regcoef.only=FALSE, ...)
## S3 method for class 'lrm':
Varcov(object, regcoef.only=FALSE, ...)
## S3 method for class 'ols':
Varcov(object, regcoef.only=FALSE, ...)
## S3 method for class 'psm':
Varcov(object, regcoef.only=FALSE, ...)

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 'glmD':
oos.loglik(fit, lp, y, ...)

num.intercepts(fit)

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)
param.order(at, term.order)
```

```
Penalty.matrix(at, X)
Penalty.setup(at, penalty)

lrtest(fit1, fit2)
## S3 method for class 'lrtest':
print(x, ...)

univarLR(fit)

Newlabels(fit, ...)
Newlevels(fit, ...)
## S3 method for class 'Design':
Newlabels(fit, labels, ...)
## S3 method for class 'Design':
Newlevels(fit, levels, ...)

DesignFit(fit)  # fit from glm, lm, etc.,then use anova etc. on result
```

Arguments

fit	result of a fitting function
object	result of a fitting function
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.
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.
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	set to "direct" to return lists of indexes of directly related factors only (those in interactions with the predictor)

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
x	a result of lrtest
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").
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.
...	other arguments; for Varcov the first argument is the fit object

Value

Varcov returns a variance-covariance matrix, and num.intercepts returns an integer with the number of intercepts in the model. 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 returns 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. DesignFit returns the original object but with oldClass of "Design" and with a new attribute "fitFunction" containing the original vector of classes.

See Also

Design, fastbw, anova.Design, summary.lm, summary.glm, datadist, vif, bootcov

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)
```

`Design` *Design 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.Design`, `summary.Design`, `predict.Design`, `plot.Design`, `survplot`, `fastbw`, `validate`, `calibrate`, `specs.Design`, `which.influence`, `latex.Design`, `nomogram.Design`, `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, called by a modified version of `model.frame.default` to store 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`. In addition to `model.frame.default` being replaced by a modified version, `[.` and `[.factor` are replaced by versions which carry along the `label` attribute of a variable. In this way, when an `na.action` function is called to subset out NAs, labels are still defined for variables in the model.

Usage

```
Design(mf, 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.Design, plot.Design,
# survplot.Design, 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 Design.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(f, x1=NA)
# latex(f)
# nomogram(f)
```

Arguments

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
f.harrell@vanderbilt.edu

See Also

Design.trans, Design.Misc, cph, lrm, ols, specs.Design, anova.Design, summary.Design, predict.Design, gendata, plot.Design, fastbw, validate, calibrate, which.influence, latex, latex.Design, model.frame.default, datadist, describe, nomogram, vif, dataRep

Examples

```
## Not run:
library(Design, first=TRUE)  # omit first for R
dist <- datadist(data=2)     # can omit if not using summary, 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
plot(f, age=NA, height=NA)    # 3-D plot
plot(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.Design(f)               # Use .Design 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 plot.Design 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=NA, x2=.5)         # Now you can do plot, summary
nomogram(f, interact=list(x2=c(.2,.7)))
## End(Not run)
```

`Design.trans` *Design Special Transformation Functions*

Description

This is a series of functions (`asis`, `pol`, `lsp`, `rcs`, `catg`, `scored`, `strat`, `matrx`, 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.Design`, `summary.Design`, `plot.Design`, `survplot`, `fastbw`, `validate`, `specs`, `which.influence`, `nomogram.Design` and `latex.Design` use these attributes to automate certain analyses (e.g., automatic tests of linearity for each predictor are done by `anova.Design`). Many of the functions are called implicitly. Some S functions such as `ns` derive data-dependent transformations that are not "remembered" when predicted values are later computed, so the predictions will be incorrect. The functions listed here solve that problem.

`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), `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).

In the list below, functions `asis` through `strat` can have arguments `x, parms, label, name` except that `parms` does not apply to `asis, matrx, strat`.

Usage

```
asis(x, parms, label, name)
matrx(x, label, name)
pol(x, parms, label, name)
lsp(x, parms, label, name)
rcs(x, parms, label, name)
catg(x, parms, label, name)
scored(x, parms, label, name)
strat(x, label, name)
x1 %ia% x2
```

Arguments

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. 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
...	a variety of things

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

rcspline.eval, rcspline.restate, Design, cph, lrm, ols, datadist

Examples

```
## Not run:
options(knots=4, poly.degree=2)
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 a Design 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
## 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.Design` is the method for creating S functions to compute X beta, based on a model fitted with `Design` in effect. Like `latex.Design`, `Function.Design` simplifies restricted cubic spline functions and factors out terms in second-order interactions. `Function.Design` 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.

Usage

```
## S3 method for class 'Design':
Function(object, intercept=NULL, digits=max(8,
```

```
.Options$digits), ...)
## S3 method for class 'cph':
Function(object, intercept=-object$center, ...)

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

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

Arguments

object	a fit created with Design 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
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.Design 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.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

latex.Design, Function.transcan, predict.Design, Design, Design.trans

Examples

```
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)
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:
# 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)
```

`Overview` *Overview of Design Library*

Description

Design does regression modeling, testing, estimation, validation, graphics, prediction, and typesetting by storing enhanced model design attributes in the fit.

Design is a collection of about 180 functions that assist and streamline modeling, especially for biostatistical and epidemiologic applications. It also contains new 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. Design 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, and the Buckley-James model. You should install the Hmisc library before using Design, as a few of Design's options use Hmisc functions, and Hmisc has several functions useful for data analysis (especially data reduction and imputation).

Details

To make use of automatic typesetting features you must have LaTeX or one of its variants installed.

Some aspects of Design (e.g., `latex`) will not work correctly if `options(contrasts=)` other than `c("contr.treatment", "contr.poly")` are used.

Design 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)
- Cox model
- Parametric survival models in the accelerated failure time class
- Buckley-James least-squares linear regression model with possibly right-censored responses
- 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
- Multiple imputation of repeated measures data with non- random dropout using propensity score matching (experimental, not yet functional)

Motivation

Design 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(f, cholesterol=NA) # cholesterol on x-axis, default range
    ```
- 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. Design 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)

Fitting Functions Compatible with Design

Design 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`
psm	Accelerated failure time parametric survival model	`survreg`
cph	Cox proportional hazards regression	`coxph`
bj	Buckley-James censored least squares linear model	`survreg`
glmD	Version of glm for use with Design	
glsD	Version of gls for use with Design	

Methods in Design

The following generic functions work with fits with Design 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`
plot	Plot effects of predictors	
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`
survest	Estimate survival probabilities (for `psm, cph`)	`survfit`
survplot	Plot survival curves (psm, cph)	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 Design 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 $\leq$ 10 unique values), the list of unique values is also stored. Such summaries are used by the summary.Design, plot.Design, and nomogram.Design 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 or plot.

```
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, newdata)
```

In the **Examples** section there are three detailed examples using a fitting function designed to be used with Design, 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 library'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 Design library and store predictor characteristics from datadist.

Detailed Example 3 shows some of the survival analysis capabilities of Design 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` × `sex` interaction. In the analysis below we do not make use of the linearity in age. Design 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 Design 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 Design. 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 Library

If you are using any of Design's survival analysis functions, create a file called `.Rprofile` in your working directory that contains the line `library(survival)`. That way, survival will move down the search list as Hmisc and Design are attached during your session. This will allow Hmisc and Design to override some of the survival function such as `survfit`.

Since the Design library has a `.First.lib` function, that function will be executed by the `library` command, to dynamically load the `.o` or `.obj` files. You may want to create a `.First` function such as

```
.First <- {
options(na.action = "na.delete")
# gives more info than na.omit
library(Hmisc)
library(Design)
invisible()
}
```

Published Applications of Design 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.Design`

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 Design 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 Design, along with case studies which includes the Design 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 ⟨f.harrell@vanderbilt.edu⟩. 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: `dump("myfit","/tmp/dumpdata")` will create a text file called `"dumpdata"` 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 `dump`'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

Acknowledgements

This work was supported by grants from the Agency for Health Care Policy and Research (US Public Health Service) and the Robert Wood Johnson Foundation.

Author(s)

Frank E Harrell Jr
Professor of Biostatistics
Chair, Department of Biostatistics
Vanderbilt University School of Medicine
Nashville, Tennessee
⟨f.harrell@vanderbilt.edu⟩

References

The primary resource for the Design library is *Regression Modeling Strategies* by FE Harrell (Springer-Verlag, 2001) and the web pages http://biostat.mc.vanderbilt.edu/rms and http://biostat.mc.vanderbilt.edu/s/Design.html. See also the Statistics in Medicine articles by Harrell *et al* listed below for case studies of modeling and model validation using Design. Also see the free book by Alzola and Harrell at http://biostat.mc.vanderbilt.edu.

Several datasets useful for multivariable modeling with Design are found at http://biostat.mc.vanderbilt.edu/s/data.

Examples

```
######################
# Detailed Example 1 #
######################
# May want to first invoke the Hmisc store function
# so that new variables will go into a temporary directory
set.seed(17)  # So can repeat random number sequence
n <- 500

sex    <- factor(sample(c('female','male'), n, rep=TRUE))
```

```
age    <- rnorm(n, 50, 10)
sys.bp <- rnorm(n, 120, 7)

# Use two population models, one with a systolic
# blood pressure effect and one without

L    <- ifelse(sex=='female', .1*(pmin(age,50)-50), .005*(age-50)^2)
L.bp <- L + .4*(pmax(sys.bp,120)-120)

dz    <- ifelse(runif(n) <= plogis(L),    1, 0)
dz.bp <- ifelse(runif(n) <= plogis(L.bp), 1, 0)

# Use summary.formula in the Hmisc library to summarize the
# data one predictor at a time

s <- summary(dz.bp ~ age + sex + sys.bp)
options(digits=3)
print(s)
plot(s)

plsmo(age, dz, group=sex, fun=qlogis, ylim=c(-3,3))
plsmo(age, L,  group=sex, method='raw', add=TRUE, prefix='True', trim=0)
title('Lowess-smoothed Estimates with True Regression Functions')

dd <- datadist(age, sex, sys.bp)
options(datadist='dd')
# can also do: dd <- datadist(dd, newvar)

f <- lrm(dz ~ rcs(age,5)*sex, x=TRUE, y=TRUE)
f
# x=TRUE, y=TRUE for pentrace

fpred <- Function(f)
fpred
fpred(age=30, sex=levels(sex))

anova(f)

p <- plot(f, age=NA, sex=NA, conf.int=FALSE, ylim=c(-3,3))
datadensity(p, age, sex)
scat1d(age)

plsmo(age, L, group=sex, method='raw', add=TRUE, prefix='True', trim=0)
title('Spline Fits with True Regression Functions')

f.bp <- lrm(dz.bp ~ rcs(age,5)*sex + rcs(sys.bp,5))

for(method in c('persp','image'))
  p <- plot(f.bp, age=NA, sys.bp=NA, method=method)
# Legend(p)   # NOTE: Needs subplot - not in R

cat('Doing 25 bootstrap repetitions to validate model\n')
validate(f, B=25)   # in practice try to use 150
```

```
cat('Doing 25 bootstrap reps to check model calibration\n')
cal <- calibrate(f, B=25)   # use 150 in practice
plot(cal)
title('Calibration of Unpenalized Model')

p <- if(.R.) pentrace(f, penalty=c(.009,.009903,.02,.2,.5,1)) else
             pentrace(f, penalty=1, method='optimize')

f <- update(f, penalty=p$penalty)
f
specs(f,long=TRUE)
edf <- effective.df(f)

p <- plot(f, age=NA, sex=NA, conf.int=FALSE, ylim=c(-3,3))
datadensity(p, age, sex)
scat1d(age)

plsmo(age, L, group=sex, method='raw', add=TRUE, prefix='True', trim=0)
title('Penalized Spline Fits with True Regression Functions')

options(digits=3)
s <- summary(f)
s
plot(s)

s <- summary(f, sex='male')
plot(s)

fpred <- Function(f)
fpred
fpred(age=30, sex=levels(sex))
sascode(fpred)

cat('Doing 40 bootstrap reps to validate penalized model\n')
validate(f, B=40)

cat('Doing 40 bootstrap reps to check penalized model calibration\n')
cal <- calibrate(f, B=40)
plot(cal)
title('Calibration of Penalized Model')

nomogram(f.bp, fun=plogis,
         funlabel='Prob(dz)',
         fun.at=c(.15,.2,.3,.4,.5,.6,.7,.8,.9,.95,.975),
         fun.side=c(1,3,1,3,1,3,1,3,1,3,1))
options(datadist=NULL)

#####################
#Detailed Example 2 #
#####################
# Simulate the data.
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)
cholesterol[1:3] <- NA   # 3 missings, at random

ddist <- datadist(cholesterol, treat, num.diseases,
                  age, weight, sex)
# Could have used ddist <- datadist(data.frame.name)
options(datadist="ddist") # defines data dist. to Design
cholesterol <- impute(cholesterol) # see impute in Hmisc library
# impute, describe, and several other basic functions are
# distributed as part of the Hmisc library

fit <- lrm(y ~ treat*log(cholesterol - 10) +
           scored(num.diseases) +  rcs(age))

describe(y ~ treat + scored(num.diseases) + rcs(age))
# or use describe(formula(fit)) for all variables used in fit
# describe function (in Hmisc) gets simple statistics on variables
#fit <- robcov(fit) # Would make all statistics which follow
                    # use a robust covariance matrix
                    # would need x=TRUE, y=TRUE in lrm
specs(fit) # Describe the design characteristics
a <- anova(fit)
print(a, which='subscripts')          # print which parameters being tested
plot(anova(fit)) # Depict Wald statistics graphically
anova(fit, treat, cholesterol) # Test these 2 by themselves
summary(fit) # Estimate effects using default ranges
plot(summary(fit)) # Graphical display of effects with C.L.
summary(fit, treat="b", age=60)
# Specify reference cell and adjustment val

summary(fit, age=c(50,70)) # Estimate effect of increasing age from
                           # 50 to 70
summary(fit, age=c(50,60,70)) # Increase age from 50 to 70,
                              # adjust to 60 when estimating
```

```
                                # effects of other factors
# If had not defined datadist, would have to define
# ranges for all var.

# Estimate and test treatment (b-a) effect averaged
# over 3 cholesterols
contrast(fit, list(treat='b',cholesterol=c(150,200,250)),
              list(treat='a',cholesterol=c(150,200,250)),
         type='average')
# Remove type='average' to get 3 separate contrasts for b-a

# Plot effects.  plot(fit) plots effects of all predictors,
# showing values used for interacting factors as subtitles
# The ref.zero parameter is helpful for showing effects of
# predictors on a common scale for comparison of strength
plot(fit, ref.zero=TRUE, ylim=c(-2,2))

plot(fit, age=seq(20,80,length=100), treat=NA, conf.int=FALSE)
# Plots relationship between age and log
# odds, separate curve for each treat, no C.I.
plot(fit, age=NA, cholesterol=NA)
# 3-dimensional perspective plot for age, cholesterol, and
# log odds using default ranges for both variables
plot(fit, num.diseases=NA, fun=function(x) 1/(1+exp(-x)),  #or fun=plogis
     ylab="Prob", conf.int=.9)
# Plot estimated probabilities instead of log odds
# Again, if no datadist were defined, would have to
# tell plot all limits
logit <- predict(fit, expand.grid(treat="b",num.diseases=1:3,
                 age=c(20,40,60),
                 cholesterol=seq(100,300,length=10)))
#logit <- predict(fit, gendata(fit, nobs=12))
# Interactively specify 12 predictor combinations using UNIX
# For UNIX or Windows, generate 9 combinations with other variables
# set to defaults, get predicted values
logit <- predict(fit, gendata(fit, age=c(20,40,60),
                 treat=c('a','b','c')))

# Since age doesn't interact with anything, we can quickly and
# interactively try various transformations of age,
# taking the spline function of age as the gold standard. We are
# seeking a linearizing transformation.  Here age is linear in the
# population so this is not very productive.  Also, if we simplify the
# model the total degrees of freedom will be too small and
# confidence limits too narrow

ag <- 10:80
logit <- predict(fit, expand.grid(treat="a",
                 num.diseases=0, age=ag,
                 cholesterol=median(cholesterol)),
                 type="terms")[,"age"]
# Note: if age interacted with anything, this would be the age
#               "main effect" ignoring interaction terms
```

```
# Could also use
#   logit <- plot(f, age=ag, ...)$x.xbeta[,2]
# which allows evaluation of the shape for any level
# of interacting factors.  When age does not interact with
# anything, the result from
# predict(f, ..., type="terms") would equal the result from
# plot if all other terms were ignored
# Could also use
#   logit <- predict(fit, gendata(fit, age=ag, cholesterol=median...))

plot(ag^.5, logit)  # try square root vs. spline transform.
plot(ag^1.5, logit) # try 1.5 power

# w <- latex(fit)  # invokes latex.lrm, creates fit.tex
# print(w)         # display or print model on screen

# Draw a nomogram for the model fit
nomogram(fit, fun=plogis, funlabel="Prob[Y=1]")

# Compose S function to evaluate linear predictors from fit
g <- Function(fit)
g(treat='b', cholesterol=260, age=50)
# Leave num.diseases at reference value

# Use the Hmisc dataRep function to summarize sample
# sizes for subjects as cross-classified on 2 key
# predictors
drep <- dataRep(~ roundN(age,10) + num.diseases)
print(drep, long=TRUE)

# Some approaches to making a plot showing how
# predicted values vary with a continuous predictor
# on the x-axis, with two other predictors varying

fit <- lrm(y ~ log(cholesterol - 10) +
           num.diseases + rcs(age) + rcs(weight) + sex)

combos <- gendata(fit, age=10:100,
                  cholesterol=c(170,200,230),
                  weight=c(150,200,250))
# num.diseases, sex not specified -> set to mode
# can also used expand.grid

combos$pred <- predict(fit, combos)
library(lattice)
xyplot(pred ~ age | cholesterol*weight, data=combos)
xYplot(pred ~ age | cholesterol, groups=weight,
       data=combos, type='l') # in Hmisc
xYplot(pred ~ age, groups=interaction(cholesterol,weight),
       data=combos, type='l')

# Can also do this with plot.Design but a single
# plot may be busy:
```

```
ch <- c(170, 200, 230)
plot(fit, age=NA, cholesterol=ch, weight=150,
     conf.int=FALSE)
plot(fit, age=NA, cholesterol=ch, weight=200,
     conf.int=FALSE, add=TRUE)
plot(fit, age=NA, cholesterol=ch, weight=250,
     conf.int=FALSE, add=TRUE)

#Here we use plot.Design to make 9 separate plots, with CLs
d <- expand.grid(cholesterol=c(170,200,230),
                 weight=c(150,200,250))
for(i in 1:nrow(d)) {
  plot(fit, age=NA, cholesterol=d$cholesterol[i],
       weight=d$weight[i])
  title(paste('Chol=',format(d$cholesterol[i]),' ',
              'Wt=',format(d$weight[i]),sep=''))
}
options(datadist=NULL)

######################
# Detailed Example 3 #
######################
n <- 2000
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"
age.dec <- cut2(age, g=10, levels.mean=TRUE)
dd <- datadist(age, sex, age.dec)
options(datadist='dd')
Srv <- Surv(t,e)

# Fit a model that doesn't assume anything except
# that deciles are adequate representations of age
f <- cph(Srv ~ strat(age.dec)+strat(sex), surv=TRUE)
# surv=TRUE speeds up computations, and confidence limits when
# there are no covariables are still accurate.

# Plot log(-log 3-year survival probability) vs. mean age
# within age deciles and vs. sex
plot(f, age.dec=NA, sex=NA, time=3,
     loglog=TRUE, val.lev=TRUE, ylim=c(-5,-1))

# Fit a model assuming proportional hazards for age and
# absence of age x sex interaction
```

```
f <- cph(Srv ~ rcs(age,4)+strat(sex), surv=TRUE)
survplot(f, sex=NA, n.risk=TRUE)
# Add ,age=60 after sex=NA to tell survplot use age=60
# Validate measures of model performance using the bootstrap
# First must add data (design matrix and Srv) to fit object
f <- update(f, x=TRUE, y=TRUE)
validate(f, B=10, dxy=TRUE, u=5)  # use t=5 for Dxy (only)
# Use B=150 in practice
# Validate model for accuracy of predicting survival at t=1
# Get Kaplan-Meier estimates by divided subjects into groups
# of size 200 (for other values of u must put time.inc=u in
# call to cph)
cal <- calibrate(f, B=10, u=1, m=200)  # B=150 in practice
plot(cal)
# Check proportional hazards assumption for age terms
z <- cox.zph(f, 'identity')
print(z); plot(z)

# Re-fit this model without storing underlying survival
# curves for reference groups, but storing raw data with
# the fit (could also use f <- update(f, surv=FALSE, x=TRUE, y=TRUE))
f <- cph(Srv ~ rcs(age,4)+strat(sex), x=TRUE, y=TRUE)
# Get accurate C.L. for any age
# Note: for evaluating shape of regression, we would not ordinarily
# bother to get 3-year survival probabilities - would just use X * beta
# We do so here to use same scale as nonparametric estimates
f
anova(f)
ages <- seq(20, 80, by=4)   # Evaluate at fewer points. Default is 100
                            # For exact C.L. formula n=100 -> much memory
plot(f, age=ages, sex=NA, time=3, loglog=TRUE, ylim=c(-5,-1))

# Fit a model assuming proportional hazards for age but
# allowing for general interaction between age and sex
f <- cph(Srv ~ rcs(age,4)*strat(sex), x=TRUE, y=TRUE)
anova(f)
ages <- seq(20, 80, by=6)
# Still fewer points - more parameters in model

# Plot 3-year survival probability (log-log and untransformed)
# vs. age and sex, obtaining accurate confidence limits
plot(f, age=ages, sex=NA, time=3, loglog=TRUE, ylim=c(-5,-1))
plot(f, age=ages, sex=NA, time=3)
# Having x=TRUE, y=TRUE in fit also allows computation of influence stats
r <- resid(f, "dfbetas")
which.influence(f)
# Use survest to estimate 3-year survival probability and
# confidence limits for selected subjects
survest(f, expand.grid(age=c(20,40,60), sex=c('Female','Male')),
        times=c(2,4,6), conf.int=.95)

# Create an S function srv that computes fitted
# survival probabilities on demand, for non-interaction model
```

```
f <- cph(Srv ~ rcs(age,4)+strat(sex), surv=TRUE)
srv <- Survival(f)
# Define functions to compute 3-year estimates as a function of
# the linear predictors (X*Beta)
surv.f <- function(lp) srv(3, lp, stratum="sex=Female")
surv.m <- function(lp) srv(3, lp, stratum="sex=Male")
# Create a function that computes quantiles of survival time
# on demand
quant <- Quantile(f)
# Define functions to compute median survival time
med.f <- function(lp) quant(.5, lp, stratum="sex=Female")
med.m <- function(lp) quant(.5, lp, stratum="sex=Male")
# Draw a nomogram to compute several types of predicted values
nomogram(f, fun=list(surv.m, surv.f, med.m, med.f),
         funlabel=c("S(3 | Male)","S(3 | Female)",
                    "Median (Male)","Median (Female)"),
         fun.at=list(c(.8,.9,.95,.98,.99),c(.1,.3,.5,.7,.8,.9,.95,.98),
                     c(8,12),c(1,2,4,8,12)))
options(datadist=NULL)

#########################################################
# Simple examples using small datasets for checking    #
# calculations across different systems in which random#
# number generators cannot be synchronized.            #
#########################################################

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)
dd <- datadist(x1,x2,x3)
options(datadist='dd')
f  <- lrm(y ~ rcs(x1,3) + x2 + x3)
f
specs(f, TRUE)
anova(f)
anova(f, x1, x2)
plot(anova(f))
s <- summary(f)
s
plot(s, log=TRUE)
par(mfrow=c(2,2))
plot(f)
par(mfrow=c(1,1))
nomogram(f)
g <- Function(f)
g(11,7,'1')
contrast(f, list(x1=11,x2=7,x3='1'), list(x1=10,x2=6,x3='2'))
fastbw(f)
gendata(f, x1=1:5)
# w <- latex(f)

f <- update(f, x=TRUE,y=TRUE)
```

```
which.influence(f)
residuals(f,'gof')
robcov(f)$var
validate(f, B=10)
cal <- calibrate(f, B=10)
plot(cal)

f <- ols(y ~ rcs(x1,3) + x2 + x3, x=TRUE, y=TRUE)
anova(f)
anova(f, x1, x2)
plot(anova(f))
s <- summary(f)
s
plot(s, log=TRUE)
par(mfrow=c(2,2))
plot(f)
par(mfrow=c(1,1))
nomogram(f)
g <- Function(f)
g(11,7,'1')
contrast(f, list(x1=11,x2=7,x3='1'), list(x1=10,x2=6,x3='2'))
fastbw(f)
gendata(f, x1=1:5)
# w <- latex(f)

f <- update(f, x=TRUE,y=TRUE)
which.influence(f)
residuals(f,'dfbetas')
robcov(f)$var
validate(f, B=10)
cal <- calibrate(f, B=10)
plot(cal)

S <- Surv(c(1,4,2,3,5,8,6,7,20,18,19,9,12,10,11,13,16,14,15,17))
survplot(survfit(S ~ x3))
f <- psm(S ~ rcs(x1,3)+x2+x3, x=TRUE,y=TRUE)
f
# NOTE: LR chi-sq of 39.67 disagrees with that from old survreg
# and old psm (77.65); suspect were also testing sigma=1

for(w in c('survival','hazard'))
 print(survest(f, data.frame(x1=7,x2=3,x3='1'),
       times=c(5,7), conf.int=.95, what=w))
# S-Plus 2000 using old survival library:
#  S(t):.925 .684 SE:0.729 0.556 Hazard:0.0734 0.255

plot(f, x1=NA, time=5)
f$var
set.seed(3)
# robcov(f)$var when score residuals implemented
bootcov(f, B=30)$var
validate(f, B=10)
cal <- calibrate(f, u=5, B=10, m=10)
```

```
plot(cal)
r <- resid(f)
survplot(r)

f <- cph(S ~ rcs(x1,3)+x2+x3, x=TRUE,y=TRUE,surv=TRUE,time.inc=5)
f
plot(f, x1=NA, time=5)
robcov(f)$var
bootcov(f, B=10)
validate(f, B=10)
cal <- calibrate(f, u=5, B=10, m=10)
survplot(f, x1=c(2,19))
options(datadist=NULL)
```

`Surv.accessors` *Accessors for survivals Surv object.*

Description

These function allow code to access the components of a survival Surv object in a implementation independant way.

Usage

```
Surv.event(surv)
Surv.strata(surv)
Surv.time(surv)
```

Arguments

`surv`	a survival object

Value

returns the part of the object requested

Author(s)

Charles Dupont

See Also

`Surv`

anova.Design *Analysis of Variance (Wald 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. print.anova.Design is the printing method. text.anova.Design is the text method for inserting anova tables on graphs. plot.anova.Design draws dot charts depicting the importance of variables in the model, as measured by Wald 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.Design 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.

Usage

```
## S3 method for class 'Design':
anova(object, ..., main.effect=FALSE, tol=1e-9,
      test=c('F','Chisq'), ss=TRUE)

## S3 method for class 'anova.Design':
print(x, which=c('none','subscripts','names','dots'), ...)

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

## S3 method for class 'anova.Design':
text(x, at, cex=.5, font=2, ...)

## S3 method for class 'anova.Design':
latex(object, title, psmall=TRUE,
      dec.chisq=2, dec.F=2, dec.ss=NA, dec.ms=NA, dec.P=4, ...)
```

Arguments

object	a Design fit object. object must allow Varcov 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 Wald 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. Can be optional graphical parameters to send to text or dotchart2, or other parameters to send to latex.default. Ignored for print.
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.
ss	For an ols fit, set ss=FALSE to suppress printing partial sums of squares, mean squares, and the Error SS and MS.
x	for print,plot,text is the result of anova.
which	If which is not "none" (the default), print.anova.Design 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.
at	for text is a list containing the x- and y-coordinates for the upper left corner of the anova table to be drawn on an existing plot, e.g. at=locator(1)
cex	character expansion size for text.anova.Design
font	font for text.anova.Design. Default is 2 (usually Courier).
what	what type of statistic to plot. The default is the Wald χ^2 statistic for each factor (adding in the effect of higher-ordered factors containing that factor) minus its degrees of freedom. The last three choice 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

pl	set to FALSE to suppress plotting. This is useful when you only wish to analyze the vector of statistics returned.
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.
psmall	The default is psmall=TRUE, which causes P<0.00005 to print as <0.0001. Set to FALSE to print as 0.0000.
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

Details

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

Value

anova.Design returns a matrix of class anova.Design containing factors as rows and χ^2, d.f., and P-values as columns (or d.f., partial SS, MS, F, P). plot.anova.Design 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, text uses tempfile to get a temporary Unix file name, sink, and unix (to remove the temporary file). 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
f.harrell@vanderbilt.edu

See Also

Design, Design.Misc, lrtest, Design.trans, summary.Design, solvet, text, locator, dotchart2, latex, Dotplot, anova.lm, contrast.Design

Examples

```
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))
anova(fit)                       # Test all factors
anova(fit, treat, cholesterol)   # Test these 2 by themselves
                                 # to get their pooled effects
g <- lrm(y ~ treat*rcs(age))
dd <- datadist(treat, num.diseases, age, cholesterol)
options(datadist='dd')
plot(g, age=NA, treat="b")
s <- anova(g)
print(s)
#p <- locator(1)                  # click mouse at upper left corner of table
p <- list(x=32,y=2.1)
text(s, at=p)                    # add anova table to regression plot
plot(s)                          # new plot - dot chart of chisq-d.f.
# latex(s)                       # nice printout - creates anova.g.tex
options(datdist=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
text(an, at=list(x=1.5,y=.6))      # add anova table to plot
plot(an)                           # new plot - dot chart of chisq-d.f.
# latex(an)                        # nice printout - creates anova.f.tex

# 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.  We rank the negative of the adjusted
# chi-squares so that a rank of 1 is assigned to the highest.
# It is important to tell plot.anova.Design not to sort the results,
# or every bootstrap replication would have ranks of 1,2,3 for the stats.

mydata <- data.frame(x1=runif(200), x2=runif(200),
                     sex=factor(sample(c('female','male'),200,TRUE)))
set.seed(9)  # so can reproduce example
mydata$y <- ifelse(runif(200)<=plogis(mydata$x1-.5 + .5*(mydata$x2-.5) +
                   .5*(mydata$sex=='male')),1,0)

if(.R.) {
library(boot)
b <- boot(mydata, function(data, i, ...) rank(-plot(anova(
                lrm(y ~ rcs(x1,4)+pol(x2,2)+sex,data,subset=i)),
                sort='none', pl=FALSE)),
                R=25)  # should really do R=500 but will take a while
Rank <- b$t0
lim <- t(apply(b$t, 2, quantile, probs=c(.025,.975)))
} else {
b <- bootstrap(mydata, rank(-plot(anova(
                lrm(y ~ rcs(x1,4)+pol(x2,2)+sex,mydata)), sort='none', pl=FALSE)),
               B=25)  # should really do B=500 but will take a while
Rank <- b$observed
lim <- limits.emp(b)[,c(1,4)]  # get 0.025 and 0.975 quantiles
}

# Use the Hmisc Dotplot function to display ranks and their confidence
# intervals.  Sort the categories by descending adj. chi-square, for ranks
original.chisq <- plot(anova(lrm(y ~ rcs(x1,4)+pol(x2,2)+sex,data=mydata)),
                       sort='none', pl=FALSE)
predictor <- as.factor(names(original.chisq))
predictor <- reorder.factor(predictor, -original.chisq)
```

```
Dotplot(predictor ~ Cbind(Rank, lim), pch=3, xlab='Rank',
                main=if(.R.) expression(paste(
'Ranks and 0.95 Confidence Limits for ',chi^2,' - d.f.')) else
'Ranks and 0.95 Confidence Limits for Chi-square - d.f.')
```

---

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 Design 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.

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=formula(data), data, 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, ...)

## 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,pr=FALSE,
                 dxy=TRUE, 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.
dxy	set to FALSE to prevent Somers' D_{xy} from being computed by validate (saves time for very large datasets)
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
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	
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 Design 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". Here sigma is the estimate of the residual standard deviation.

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
f.harrell@vanderbilt.edu

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

Design, psm, survreg, cph, Surv, na.delete, na.detail.response, datadist, rcorr.cens.

Examples

```
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")

f <- bj(Surv(ftime, stroke) ~ rcs(age,5) + 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(f, age=NA, hospital=NA)  # needs datadist since no explicit age,hosp.
coef(f)               # look at regression coefficients
coef(psm(Surv(ftime, stroke) ~ rcs(age,5) + 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(survfit(r), conf='none')
                      # plot Kaplan-Meier estimate of
                      # survival function of standardized residuals
survplot(survfit(r ~ cut2(age, g=2)), conf='none')
                      # may desire both strata to be n(0,1)
options(datadist=NULL)
```

---

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 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 nonparametric 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.

Usage

```
bootcov(fit, cluster, B=200, fitter,
        coef.reps=FALSE, loglik=coef.reps,
        pr=FALSE, maxit=15, group, stat=NULL)

bootplot(obj, which, X,
         conf.int=c(.9,.95,.99),
         what=c('density','qqnorm'),
         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.Design or contrast.Design. 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.
coef.reps	set to TRUE if you want to store a matrix of all bootstrap regression coefficient estimates in the returned component boot.Coef. For models set loglik=FALSE to get coef.reps=TRUE to work.
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 Design.Misc). 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.
maxit	maximum number of iterations, to pass to fitter
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.
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
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 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).

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.

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
⟨f.harrell@vanderbilt.edu⟩

Bill Pikounis
Biometrics Research Department
Merck Research Laboratories
⟨v_bill_pikounis@merck.com⟩

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

robcov, sample, Design, lm.fit, lrm.fit, coxph.fit, survreg.fit, predab.resample, Design.Misc, predict.Design, gendata, contrast.Design

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, coef.reps=TRUE)
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(1)
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)  # 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(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)
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, coef.reps=TRUE, pr=TRUE)   # 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

# 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 <- plot(f, age=ages, sex='female')
w <- p$x.xbeta
# Save curve coordinates for later automatic
# labeling using labcurve in the Hmisc library
curves <- vector('list',8)
curves[[1]] <- list(x=w[,1],y=w[,3])
curves[[2]] <- list(x=w[,1],y=w[,4])

# Add pointwise normal-distribution confidence
# bands using unconditional variance-covariance
# matrix from the 500 bootstrap reps
p <- plot(b, age=ages, sex='female', add=TRUE, lty=3)
```

```
w <- p$x.xbeta
curves[[3]] <- list(x=w[,1],y=w[,3])
curves[[4]] <- list(x=w[,1],y=w[,4])

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)

# 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.Design 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)
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)
set.seed(3)
f <- bootcov(f, stat='C')
set.seed(3)
g <- bootcov(g, stat='C')
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)))
## End(Not run)
```

`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 (for survival models) or on nonparametric smoothers (for other models). There are calibration functions for Cox (`cph`), parametric survival models (`psm`), binary and ordinal logistic models (`lrm`) and ordinary least squares (`ols`). For survival models, "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. For logistic and linear models, a nonparametric calibration curve is estimated over a sequence of predicted values. The fit must have specified `x=TRUE, y=TRUE`. The `print` and `plot` methods for `lrm` and `ols` models (which use `calibrate.default`) 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, the second, third, and fourth invocations of `calibrate` are, respectively, for `ols` and `lrm`, `cph`, and `psm`. The first and second `plot` invocation are respectively for `lrm` and `ols` fits or all other fits.

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, pr=FALSE, kint, smoother="lowess", ...)
## S3 method for class 'cph':
calibrate(fit, method="boot", u, m=150, cuts, B=40,
  bw=FALSE, rule="aic", type="residual", sls=0.05, aics=0,
  pr=FALSE, what="observed-predicted", tol=1e-12, ...)
## S3 method for class 'psm':
calibrate(fit, method="boot", u, m=150, cuts, B=40,
  bw=FALSE,rule="aic",
  type="residual",sls=.05,aics=0,
  pr=FALSE,what="observed-predicted",tol=1e-12, maxiter=15,
  rel.tolerance=1e-5, ...)

## S3 method for class 'calibrate':
print(x, ...)
## S3 method for class 'calibrate.default':
print(x, ...)

## S3 method for class 'calibrate':
plot(x, xlab, ylab, subtitles=TRUE, conf.int=TRUE,
...)

## S3 method for class 'calibrate.default':
plot(x, xlab, ylab, xlim, ylim,
```

```
legend=TRUE, subtitles=TRUE, ...)
```

Arguments

fit	a fit from ols, lrm, cph or psm
x	an object created by calibrate
method	
B	
bw	
rule	
type	
sls	
aics	see validate
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)
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)
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.
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).
...	other arguments to pass to predab.resample, such as 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
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.

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".

Side Effects

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

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

See Also

validate, predab.resample, groupkm, errbar, cph, psm, lowess

Examples

```
set.seed(1)
d.time <- rexp(200)
x1 <- runif(200)
x2 <- factor(sample(c('a','b','c'),200,TRUE))
f <- cph(Surv(d.time) ~ pol(x1,2)*x2, x=TRUE, y=TRUE, surv=TRUE, time.inc=2)
#or f <- psm(S ~ ...)
cal <- calibrate(f, u=2, m=50, B=20)  # usually B=200 or 300
plot(cal)

y <- sample(0:2, 200, TRUE)
```

```
x1 <- runif(200)
x2 <- runif(200)
x3 <- runif(200)
x4 <- runif(200)
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.Design` *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 Design, 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.Design` can be used by the `bootplot` and `confplot` functions to obtain bootstrap nonparametric confidence intervals for contrasts.

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).

When more than one contrast is computed, the list created by `contrast.Design` is suitable for plotting (with error bars or bands) with `xYplot` or `Dotplot` (see the last example).

Usage

```
contrast(fit, ...)
## S3 method for class 'Design':
contrast(fit, a, b, cnames=NULL,
         type=c("individual", "average"),
         weights="equal", conf.int=0.95, ...)

## S3 method for class 'contrast.Design':
print(x, X=FALSE, fun=function(u)u, ...)
```

Arguments

fit	a fit of class "Design"
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.
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.
cnames	vector of character strings naming the contrasts when type="individual". Usually cnames is not necessary as contrast.Design 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))
type	set type="average" to average the individual contrasts (e.g., to obtain a Type II or III contrast)
weights	a numeric vector, used when type="average", to obtain weighted contrasts
conf.int	confidence level for confidence intervals for the contrasts
...	unused
x	result of contrast
X	set X=TRUE to print design matrix used in computing the contrasts (or the average contrast)
fun	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.

Value

a list of class "contrast.Design" containing the elements Contrast, SE, Z, var, df.residual Lower, Upper, Pvalue, X, cnames, 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, and contrast names (or NULL).

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu

See Also

predict.Design, gendata, bootcov, summary.Design, anova.Design, plot.Design

Examples

```
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)
# 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))

# For a model containing two treatments, centers, and treatment
# x center interaction, get 0.95 confidence intervals separately
# by cente
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, coef.reps=TRUE)
bootplot(b, X=k$X)    # bootstrap distribution and CL

# 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))
## 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
plot(f, age=NA, treat=NA, sex='female')
plot(f, age=NA, treat=NA, 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))
xYplot(Cbind(Contrast, Lower, Upper) ~ age | sex, data=w,
       ylab='Drug - Placebo')
xYplot(Cbind(Contrast, Lower, Upper) ~ age, groups=sex, data=w,
       ylab='Drug - Placebo', method='alt bars')
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=if(.R.) parent.frame() else sys.parent(),
    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,
    eps=1e-4, init, iter.max=10, tol=1e-9, surv=FALSE, time.inc,
    type, vartype, conf.type, ...)

## 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, ...)
```

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 Design 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.

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=Y 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
conf.type	see survfit.cph; default bases confidence limits of log -log survival.
...	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, and R2.
var	variance/covariance matrix of coefficients

linear.predictors	values of predicted X beta for observations used in fit, normalized to have overall mean zero
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
f.harrell@vanderbilt.edu

See Also

coxph, coxph.fit, Surv, residuals.cph, cox.zph, survfit.cph, survest.cph, survfit.coxph, survplot, datadist, Design, Design.trans, anova.Design, summary.Design, predict.Design, fastbw, validate, calibrate, plot.Design, specs.Design, lrm, which.influence, na.delete, na.detail.response, naresid, print.cph, latex.cph, vif, ie.setup

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
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')
```

```
Srv <- Surv(dt,e)

f <- cph(Srv ~ rcs(age,4) + sex, x=TRUE, y=TRUE)
cox.zph(f, "rank")             # tests of PH
anova(f)
plot(f, age=NA, sex=NA)      # plot age effect, 2 curves for 2 sexes
survplot(f, sex=NA)             # 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
if(.R.) plot(time, fitted(z)) else
plot(z, coverage=0.95, confidence=7, xlab="t",
     ylab="Scaled Schoenfeld Residual",ylim=c(-3,5))
lines(supsmu(time, res[,4]),lty=2)
plot(cox.zph(f,"identity"))    #Easier approach for last 6 lines
# latex(f)

f <- cph(Srv ~ 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(f, age=NA, fun=function(x) med(lp=x))          #plot median survival

# 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
f.harrell@vanderbilt.edu

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

# In S-Plus:
#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 `plot.Design`, `summary.Design`, `survplot`, and `nomogram.Design`. 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 10^{th} smallest and 10^{th} 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
f.harrell@vanderbilt.edu

See Also

`Design`, `Design.trans`, `describe`, `plot.Design`, `summary.Design`

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(f, x1=200:800, x2=3)

# 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(fit, age=NA, ref.zero=TRUE, fun=exp, ylab='Age=x:Age=30 Odds Ratio')
## 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="aic", type="residual", sls=.05, aics=0, eps=1e-9, k.aic=2)

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

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. falls below 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-9.
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).
x	result of fastbw
digits	number of significant digits to print
...	ignored

Value

a list with 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
f.harrell@vanderbilt.edu

References

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

See Also

Design, ols, lrm, cph, psm, validate, solvet, Design.Misc

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)
```

---

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. 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, ...)
## S3 method for class 'Design':
gendata(fit, nobs, viewvals=FALSE,
  editor=.Options$editor, ..., factors)
## Default S3 method:
gendata(fit, ...)
```

Arguments

fit	a fit object created with Design in effect
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
editor	editor to use to edit the list of variable names to consider. Default is options(editor=) value ("xedit" is this is not specified by using.X()==TRUE.

...	predictor settings, if nobs is not given.
factors	a list containing predictor settings with their names. This is an alternative to specifying the variables separatey in ....

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 data.ed 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
f.harrell@vanderbilt.edu

See Also

predict.Design, survest.cph, survest.psm, Design.Misc, expand.grid, data.entry, page, print.datadist, edit.data.frame, plot.Design

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)
```

`glmD` *Design Version of glm*

Description

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

Usage

```
glmD(formula, family = gaussian, data = list(), weights = NULL, subset =
NULL, na.action = na.fail, start = NULL, offset = NULL, control =
glm.control(...), model = TRUE, method = "glm.fit", x = FALSE, y = TRUE,
contrasts = NULL, ...)

## S3 method for class 'glmD':
print(x, digits=4, ...)
```

Arguments

`formula`	
`family`	
`data`	
`weights`	
`subset`	
`na.action`	
`start`	
`offset`	
`control`	
`model`	
`method`	
`x`	
`y`	
`contrasts`	see `glm`; for `print`, `x` is the result of `glmD`
`...`	ignored for `print`
`digits`	number of significant digits to print

Value

a fit object like that produced by glm but with Design attributes and a class of "Design", "glmD", and "glm" or "glm.null".

See Also

glm,Design

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 <- glmD(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'))
```

---

glsD *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. glsD is a slightly enhanced version of the Pinheiro and Bates glsD function in the nlme package to make it easy to use with the Design library 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).

Usage

```
glsD(model, data, correlation, weights, subset, method, na.action,
    control, verbose, B=0, dupCluster=FALSE, pr=FALSE,
    opmeth=c('optimize','optim'))

## S3 method for class 'glsD':
print(x, digits=4, ...)
```

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.fail) causes gls to print an error message and terminate if there are any incomplete observations.
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 glsD 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
opmeth	specifies whether the optimize or the optim function is to be used for optimization
x	the result of glsD
digits	number of significant digits to print
...	ignored

Value

an object of classes glsD, Design, and gls representing the linear model fit. Generic functions such as print, plot, 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. `glsD` returns the following components not returned by `gls`: `Design`, `assign`, `formula`, `opmeth` (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).

Author(s)

Jose Pinheiro ⟨jcp@research.bell-labs.com⟩, Douglas Bates ⟨bates@stat.wisc.edu⟩, Frank Harrell ⟨f.harrell@vanderbilt.edu⟩, Patrick Aboyoun ⟨aboyoun@insightful.com⟩

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`

Examples

```
## Not run:
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')
# library(nlme)  # S-Plus: library(nlme3) or later
f <- glsD(y ~ pol(tim,2) + trt, correlation=corCAR1(form= ~tim | id),
          data=d, B=B)
f
f$var      # bootstrap variances
f$varBeta  # original variances
summary(f)
anova(f)
plot(f, tim=NA, trt=NA)
# v <- Variogram(f, form=~tim|id, data=d)
## 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-log K-M. Confidence intervals are based on log-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 survfit.km 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-log `KM`)

See Also

`survfit.km`, `errbar`, `cut2`, `Surv`, `units`

Examples

```
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
f.harrell@vanderbilt.edu

See Also

cox.zph, residuals.cph, coxph.fit, cph, coxph, Surv

Examples

```
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 function 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
f.harrell@vanderbilt.edu

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)
```

---

latex.Design *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. These routines work with the display package from statlib to display and print the formatted model fits. latexDesign is the core function which is called internally by latex.Design (which is called by latex.cph, latex.ols, etc.).

Usage

```
latex(object, title,
      file=paste(first.word(deparse(substitute(object))),'tex',sep='.'), ...)
```

Arguments

object	a fit object created by a fitting function in the Design series
title	ignored
file	name of .tex file to create, default is first word of argument object with ".tex" added. Set to "" to send LaTeX output to standard output.
...	further arguments, including

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$Design$assume.code!=9]. 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.

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 latex.Design 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

Other arguments in '...' will be passed to latex.default.

Value

a file name of class "latex"

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

latex, rcspline.restate, Design

Examples

```
## Not run:
f <- lrm(death ~ rcs(age)+sex)
w <- latex(f)
w     # displays, using e.g. xdvi
latex(f, file="")    # send LaTeX code to screen
## End(Not run)
```

---

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=paste(first.word(deparse(substitute(object))),".tex",sep=""),
      append=FALSE, surv=TRUE, maxt=FALSE, which=NULL, varnames, columns=65,
      inline=FALSE, before=if(inline)"" else "& &", dec=3,
      pretrans=TRUE, caption, ...) # for cph fit

## S3 method for class 'lrm':
latex(object, title, file, append, which, varnames,
columns, inline, before, pretrans, caption, ...) # for lrm fit

## S3 method for class 'ols':
latex(object, title, file, append, which, varnames,
columns, inline, before, pretrans, caption, ...) # ols fit

## S3 method for class 'pphsm':
latex(object, title, file, append, which=NULL, varnames,
columns, inline, before, pretrans, caption, ...) # pphsm fit

## S3 method for class 'psm':
latex(object, title, file, append, which=NULL, varnames,
columns, inline, before, pretrans, caption, ...) # psm fit
```

Arguments

object	a fit object created by a Design fitting function.
title	ignored
file	
append	see latex.default
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
caption	a character string specifying a title for the equation to be centered and typeset in bold face. Default is no title.
...	ignored

Value

the name of the created file, with class c("latex","file"). This object works with latex viewing and printing commands in Hmisc.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

latex.Design, rcspline.restate, latex.default

Examples

```
## Not run:
units(ftime) <- "Day"
f <- cph(Surv(ftime, death) ~ rcs(age)+sex, surv=TRUE, time.inc=60)
w <- latex(f)  #Interprets fitted model and makes table of S0(t)
               #for t=0,60,120,180,...   Creates file f.tex
w              #displays image, if viewer installed
latex(f,file="")   # send LaTeX code to the screen
## 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`.

Usage

```
lrm(formula, data, subset, na.action=na.delete, method="lrm.fit",
    model=FALSE, x=FALSE, y=FALSE, linear.predictors=TRUE, se.fit=FALSE,
    penalty=0, penalty.matrix, tol=1e-7,
    strata.penalty=0, var.penalty=c('simple','sandwich'),
    weights, normwt, ...)
```

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`.
`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, only 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 `Design.Misc`. If you specify `penalty.matrix` you must specify a nonzero value of `penalty` or no penalization will be done.

tol — singularity criterion (see `lrm.fit`)

strata.penalty — scalar penalty factor for the stratification factor, for the experimental `strat` variable

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.

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 `lrm.fit`.

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, and the Brier score computed with respect to $Y >$ its lowest level. Probabilities are rounded to the nearest 0.002 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 (the default, if `var.penalty="simple"`) or the sandwich-type variance - covariance matrix estimate (Gray Eq. 2.6) if `var.penalty="sandwich"`. For the latter case the simple information-matrix - based variance matrix is returned under the name `var.from.info.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

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

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, Design.trans, Design, glm, latex.lrm, residuals.lrm, na.delete, na.detail.response, naresid, pentrace, Design.Misc, vif, cr.setup, predab.resample, validate.lrm, calibrate, Mean.lrm

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
#
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'

#To use prop. odds model, avoid using a huge number of intercepts by
#grouping cholesterol into 40-tiles
ch <- cut2(cholesterol, g=40, levels.mean=TRUE) # use mean values in intervals
table(ch)
f <- lrm(ch ~ age)
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 plot.Design
#      or summary.Design would require access to the original dataset or
#      d or specifying all variable values to summary, plot, nomogram
anova(fit)
plot(fit, age=NA, sex=NA)
plot(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 ~ ..., penalty=...)

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

Description

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

Usage

```
lrm.fit(x, y, offset, initial, est, maxit=12, eps=.025,
        tol=1E-7, trace=FALSE, penalty.matrix, weights, normwt)
```

Arguments

`x`	design matrix with no column for an intercept
`y`	response vector, numeric, categorical, or character
`offset`	optional numeric vector containing an offset on the logit scale
`initial`	vector of initial parameter estimates, beginning with the intercept
`est`	indexes of `x` to fit in the model (default is all columns of `x`). Specifying `est=c(1,2,5)` causes columns 1,2, and 5 to have parameters estimated. The score vector `u` and covariance matrix `var` can be used to obtain score statistics for other columns
`maxit`	maximum no. iterations (default=`12`). Specifying `maxit=1` causes logist to compute statistics at initial estimates.
`eps`	difference in `-2 log` likelihood for declaring convergence. Default is `.025`.
`tol`	Singularity criterion. Default is 1E-7
`trace`	set to `TRUE` to print -2 log likelihood, step-halving fraction, and rank of variance matrix at each iteration
`penalty.matrix`	a self-contained ready-to-use penalty matrix - see `lrm`
`weights`	a vector (same length as `y`) of possibly fractional case weights
`normwt`	set to codeTRUE to scale `weights` so they sum to the length of `y`; useful for sample surveys as opposed to the default of frequency weighting

Value

a list with the following components:

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 chi-square, d.f., P-value, c index (area under ROC curve), Somers' D_{xy}, Goodman-Kruskal γ, and Kendall's τ_a rank correlations between predicted probabilities and observed response, the Nagelkerke R^2 index, and the Brier probability score with respect to computing the probability that $y >$ lowest level. Probabilities are rounded to the nearest 0.002 in the computations or rank correlation indexes. 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 maxiter>1)
coefficients	estimated parameters
var	estimated variance-covariance matrix (inverse of information matrix). Note that in the case of penalized estimation, var is not the improved sandwich-type estimator (which lrm does compute).
u	vector of first derivatives of log-likelihood
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.
est	vector of column numbers of X fitted (intercepts are not counted)
non.slopes	number of intercepts in model
penalty.matrix	see above

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

lrm, glm, matinv, solvet, cr.setup

Examples

```
#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), death)
```

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*

Description

Draws a partial nomogram that can be used to manually obtain predicted values from a regression model that was fitted with `Design` in effect. 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.

`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, ...)

## S3 method for class 'Design':
nomogram(fit, ..., adj.to,
         lp=TRUE, lp.at, lplabel="Linear Predictor",
         fun, fun.at, fun.lp.at,
         funlabel="Predicted Value", fun.side,
```

```
          interact=NULL, intercept=1, conf.int=FALSE,
          col.conf=c(1, if(under.unix).3 else 12),
          conf.space=c(.08,.2),
          conf.lp=c("representative","all", "none"),
          est.all=TRUE, abbrev=FALSE, minlength=4,
          maxscale=100, nint=10, label.every=1, force.label=FALSE,
          xfrac=0.35, cex.axis=0.85, cex.var=1, col.grid=FALSE,
          vnames=c("labels","names"), varname.label=TRUE,
          varname.label.sep="=",
          ia.space=.7, tck=-.009, lmgp=.4, omit=NULL, naxes,
          points.label='Points', total.points.label='Total Points',
          total.sep.page=FALSE, total.fun, verbose=FALSE)

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

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

Arguments

fit	a regression model fit that was created with `library(Design)` in effect, and (usually) with `options(datadist="object.name")` in effect.
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
x	
y	coordinates 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`.
...	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`.
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")`.
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"))`.

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).
lplabel	label for linear predictor axis. Default is "Linear Predictor".
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.
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).
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.
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.
col.conf	colors corresponding to conf.int. Use fractions for gray scale (for UNIX S-PLUS).
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.
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.

intercept	for models such as the ordinal logistic model with multiple intercepts, specifies which one to use in evaluating the linear predictor.
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.
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.
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 col.grid=1, no gray scale is used, but an ordinary line is drawn. If 0<col.grid<1, a col (gray scale) of col.grid is used to draw vertical reference lines for major axis divisions and col.grid/2 for minor divisions. The default is col.grid=FALSE, i.e., reference lines are omitted. Specifying col.grid=TRUE is the same as specifying a gray scale level of col.grid=.2 (5 for Windows S-PLUS).
vnames	By default, variable labels are used to label axes. Set vnames="names" to instead use variable names.
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
lmgp	spacing between numeric axis labels and axis (see par for mgp)

omit	vector of character strings containing names of variables for which to suppress drawing axes. Default is to show all variables.
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(loca
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.
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.
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 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
f.harrell@vanderbilt.edu

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, http://en.wikipedia.org/wiki/Nomogram.

See Also

`Design`, `plot.Design`, `plot.summary.Design`, `axis`, `pretty`, `approx`, `latex.Design`, `Design.Misc`

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)
```

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

f <- lrm(y ~ lsp(age,50)+sex*rcs(cholesterol,4)+blood.pressure)
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", xfrac=.45)
#Instead of fun.at, could have specified fun.lp.at=logit of
#sequence above - faster and slightly more accurate
nomogram(f, age=seq(10,90,by=10), xfrac=.45)
g <- lrm(y ~ sex + rcs(age,3)*rcs(cholesterol,3))
nomogram(g, interact=list(age=c(20,40,60)),
         conf.int=c(.7,.9,.95), col.conf=c(1,.5,.2))

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, dist=if(.R.)'lognormal' else 'gaussian')
med  <- Quantile(f)
surv <- Survival(f)  # This would also work if f was from cph
nomogram(f, fun=function(x) med(lp=x), funlabel="Median Survival Time")
nomogram(f, fun=list(function(x) surv(3, x), function(x) surv(6, x)),
            funlabel=c("3-Month Survival Probability",
                       "6-month Survival Probability"), xfrac=.5)

## 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 <- nom$abbrev$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
Y <- ifelse(y==0, 0, sample(1:3, length(y), TRUE))
g <- lrm(Y ~ age+rcs(cholesterol,4)*sex)
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
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),
         lmgp=.2, cex.axis=.6)
options(datadist=NULL)
```

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, 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=1e-7, 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 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 Design). 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, 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. Sigma is the penalized maximum likelihood estimate (see below).

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

Design, Design.trans, anova.Design, summary.Design, predict.Design, fastbw, validate, calibrate, plot.Design, specs.Design, cph, lrm, which.influence, lm, summary.lm, print.ols, residuals.ols, latex.ols, na.delete, na.detail.response, naresid, 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 plot.Design, summary.Design.  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)
```

---

pentrace *Trace AIC and BIC vs. Penalty*

Description

For an ordinary unpenalized fit from `lrm` 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,
         fitter, pr=FALSE, tol=1e-7,
         keep.coef=FALSE, complex.more=TRUE, verbose=FALSE, maxit=12, subset)

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` 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.
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

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
f.harrell@vanderbilt.edu

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, ols, solvet, Design.Misc, 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
if(!.R.) pentrace(f, 1, method='optimize')

# Find penalty reducing total regression d.f. effectively to 5
if(!.R.) 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.Design` *Plot Effects of Variables*

Description

Plots the effect of one or two predictors on the linear predictor or X beta scale, or on some transformation of that scale. The predictor is always plotted in its original coding on the x or y-axis. `perimeter` is a function 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.

`Legend` is a generic function for adding legends to an existing graph according to the specific plot made by `plot.Design`. The specific `Legend` method for `plot.Design` is `Legend.plot.Design`. It handles legends for `image` plots. For other plots with one or more curves, make legends using the `label.curves` parameter.

`datadensity` is a function for showing the data density (raw data) on each curve generated for curve-type plots. This is a rug plot showing the location/density of data values for the x-axis variable. If there was a second variable specified to `plot` that generated separate curves, the data density specific to each class of points is shown. This assumes that the second variable was categorical. The rug plots are drawn by `scat1d`.

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

Usage

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

## S3 method for class 'Design':
plot(x, ..., xlim, ylim, fun, xlab, ylab,
     conf.int=.95, conf.type=c('mean','individual'),
     add=FALSE, label.curves=TRUE,
     eye, theta=0, phi=15, perspArgs=NULL,
     lty, col=1, lwd=par('lwd'), lwd.conf=1, pch=1,
     adj.zero=FALSE, ref.zero=FALSE, adj.subtitle, cex.adj,
     non.slopes, time=NULL, loglog=FALSE, val.lev=FALSE,
     digits=4, log="", perim,
     method=c("persp","contour","image","dotchart","default"),
     sortdot=c('neither','ascending','descending'),
     nlevels=10, name, zlim=range(zmat,na.rm=TRUE),
     vnames=c('labels','names'), abbrev=FALSE)
# Just say plot(fit, ...)

## S3 method for class 'plot.Design':
print(x, ...)

## S3 method for class 'perimeter':
lines(x, ...)

Legend(object, ...)
## S3 method for class 'plot.Design':
Legend(object, x, y, size=c(1,1), horizontal=TRUE,
       nint=50, fun, at, zlab, ...)

## S3 method for class 'plot.Design':
datadensity(object, x1, x2, ...)
```

Arguments

fit	a fit object created with `Design()` in effect. `options(datadist="d")` must have been specified (where `d` was created by `datadist`), or it must have been in effect with `fit` was created.
...	The first variable in this list is displayed on the x-axis. Specify `x=NA` 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, 100 equally spaced values in the range are used for plotting if the range is not specified. If there is a second variable listed, and its range is `NA` or a single value, that variable is displayed on the y-axis. If the second variable's range has fewer than 40 levels, separate curves are generated for each value of the variable. Otherwise, a three dimensional perspective plot is drawn using 40 equally-spaced values of `y`. Names may be abbreviated. `plot` senses that a variable is not to be displayed by checking if the list of values for the variable is a scalar instead of a vector. 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. Due to a bug in S, the first variable mentioned may not be named `x`. This would cause the general scatterplot function `plot` to be invoked by mistake. Variables after the first or second specified to `plot` 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. Note that you can omit `variables` entirely. In that case, all non-interaction effects will be plotted automatically as if you said `plot(fit, age=NA); plot(fit, sex=NA); ...`. In this case you have no control over the settings of the variables for the x-axis, i.e., `NA` is always assumed. For a plot made up of multiple curves, these are extra graphical arguments will be passed to `key` from `Legend`. For `image` plots, these arguments are passed to `par` and have temporary effect. For `datadensity` these extra arguments are passed along to `scat1d`.
x	first variable of a pair of predictors forming a 3-d plot, to specify to `perim`. For `Legend`, is either a vector of 1 or 2 x-coordinates or a list with elements `x` and `y` each with 1 or 2 coordinates. For `method="image"` plots, 1 or 2 coordinates may be given, and for other plot types, 1 coordinate is given. A single coordinate represents the upper left corner of the legend box. For `Legend`, `x` and `y` are optional. If omitted, `locator` is used to position legends with the mouse. For `lines.perimeter`, `x` is the result of `perimeter`. For `print.plot.Design`, `x` is the result of `plot.Design`.
y	second variable of the pair for `perim`, or y-coordinates for `Legend`. 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
xlim	This parameter is seldom used, as limits are usually controlled with the variables specifications. 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.Design. 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.
fun	Function used to transform $X\beta$ and its confidence interval before plotting. For example, to transform from a logit to a probability scale, use fun=function(x)1/(1+exp(-x)) or fun=plogis, and to take the anti-log, specify fun=exp. for Legend, fun is a function for transforming tick mark labels for color or gray scale legends for method="image". For example, if plot.Design is used to make an image plot of log odds ratios, specifying fun=plogis will cause the color legend to be labeled with probability values rather than log odds.
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 (z-axis if perspective plot). 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. If fun is given, the default is "". If time is given, the default is "(time) (units) Survival Probability" or "log[-log S(time)]" depending on the loglog parameter.
conf.int	Default is .95. Specify FALSE to suppress confidence bands.
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".
add	Set to TRUE to add to an existing plot without drawing new axes. Default is FALSE. See the warning note under sortdot.
label.curves	Set 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.
eye	Argument to S persp function for defining perspective in 3-d plots. Default is (-6, -6, 9). This is for S-Plus only.
theta	
phi	
perspArgs	a list containing other named arguments to be passed to persp

lty	Vector of line types to use in plotting separate curves. Default is 1,2, ...
lwd	Vector of line widths corresponding to separate curves, default is par("lwd").
lwd.conf	scalar width of lines for confidence bands. Default is 1.
pch	symbol to use when plotting unconnected points when a categorical variable is on the x-axis or when method="dotchart". Default is 1 (open circle). See points for other values, or use the show.pch function in Hmisc.
col	S color number for displaying curves. Default is 1 (black). Specify a vector of integers to assign different colors to different curves.
adj.subtitle	Set to FALSE to suppress subtitling the graph with the list of settings of non-graphed adjustment values. Default is TRUE if <= 6 non-plotted factors.
cex.adj	cex parameter for size of adjustment settings in subtitles. Default is 0.75 times par("cex").
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	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.
non.slopes	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 non.slopes=c(0,1,0). The default is non.slopes=rep(0,k) if adj.zero=TRUE, where k is the number of intercepts in the model. If adj.zero=FALSE, the default is (1,0,0,...,0).
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.
val.lev	When plotting a categorical or strata factor with category labels that are strings of legal numeric values, set to TRUE to use these values in plotting. An ordinary axis with uniform spacing will be used rather than spacing dictated by the value labels. When val.lev=FALSE, category labels dictate how axis tick marks are made. val.lev is used typically when the variable being plotted is a categorical variable that was collapsed into intervals, with the value label for a category representing interval means or midpoints. Such variables are created for example by the cut2 function, specifying levels.mean=TRUE. For plotting a discrete numeric variable you can specify val.lev=TRUE to force plotting of the variable as if it were continuous.
digits	Controls how "adjust-to" values are plotted. The default is 4 significant digits.
log	Set log="x", "y" or "xy" to plot log scales on one or both axes.
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.

method — For 3-d plots, use method="persp" for perspective plots (persp(), the default), method="contour" to use contour(), or method="image" to use image(). Specify method="dotchart" to make a horizontal dot chart to represent predicted values associated with categorical predictors. The log argument does not apply to these plot types. You can specify method="default" to get the default behaviour. For "dotchart", the dotchart2 function in the Hmisc library is used.

sortdot — applies when method="dotchart". The default is to plot the points in the order requested for predictions. Specify method="ascending" or an abbreviation such as method="a" to sort in ascending order before plotting the dot chart. You may also specify method="descending". Unless method="neither", specifying add=TRUE may not work properly.

nlevels — number of contour levels if method="contour"

name — Instead of specifying the variable to plot on the x-axis in the variables list, you can specify one or more variables to plot 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; plotting is done as if you had said e.g. age=NA. Also, interacting factors can only be set to their reference values using this notation.

zlim — If 'type="persp"' controls the range for plottin in the z-axis. Computed by default.

vnames — applies when no x-variable is specified (i.e., when all predictors are being plotted). To override the default x-axis label in that case (variable "label" attributes) to instead use variable names, specify vnames="names".

object — an object created by plot.Design

abbrev — Set to TRUE to use the abbreviate function to abbreviate levels of categorical factors for labeling tick marks on the x-axis.

size

horizontal

nint — see image.legend

at — If fun is specified to Legend, at may be given. at is a vector of values at which to evaluate fun for drawing tick marks in the color legend. For example, if you want to show the median survival time for a log-normal survival model whereas the linear predictor (log median) was used in constructing the image plot, and if you want to place tick marks at nice median values, specify fun=exp, at=log(c(1,10,100,1000)).

zlab — label for image color axis legend. Default is from the model (e.g., "Log Odds"), but zlab will often be specified if fun was specified to plot.Design or Legend.

x1	data vector for first variable in plot (x-axis variable)
x2	data vector for second variable in plot if it was not constant (curve-generating variable)

Details

When there are no intercepts in the fitted model, plot subtracts adjustment values from each factor while computing variances for confidence limits.

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 plot.Design readies the data for persp, it 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 plot). To make the perimeter smooth, specify lowess.=TRUE to perimeter.

Specifying time will not work for Cox models with time-dependent covariables. Use survest or survfit for that purpose.

Use ps.slide, win.slide, gs.slide to set up nice defaults for plotting. These also 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 components of mgp using mgp.axis.labels(c(xvalue,yvalue)).

Note that because the generic plot method has the variable x as its first argument, you cannot explicitly specify that you want to plot the effect of a predictor named x.

Value

perimeter returns a matrix of class perimeter. This outline can be conveniently plotted by lines.perimeter. Legend.plot.Design invisibly returns the position of the legend. plot.Design invisibly returns an invisible object of class "plot.Design" with the following components (use print.plot.Design to get a nice printout of the object):

x.xbeta	data frame of values plotted. First column is sequence of x-values. If a second variable was plotted, second column is sequence of y-values. Next column is estimated $X\beta$, followed by a column of lower confidence limits and upper confidence limits. If fun was specified, these last three columns are transformed by the specified function.
adjust	character string of the form "sex=male age=50" containing settings of non-plotted factors.
curve.labels	character vector containing values of y-variable if it determined different curves on the plot. E.g., c("female","male") or c("10","20","30"). This vector is useful as an argument to the S key function if label.curves=FALSE.
plot.type	"curves" or "3d"
method	from plot.Design call

lty	vector of line types used for curves (if the plot used a few curves to represent a second variable plotted)
lwd	vector of line widths used
col	vector of color codes

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

datadist, predict.Design, contrast.Design, summary.Design, persp, Design, Design.trans, survest, survplot, Design.Misc, contour, image, labcurve, scat1d, dotchart2, mgp.axis.labels Overview, par, ps.slide, xYplot, smearingEst

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)

par(mfrow=c(2,2))
plot(fit)                # Plot effects of all 4 predictors
par(mfrow=c(1,2))
plot(fit, name=c('age','cholesterol'))   # Make 2 plots
par(mfrow=c(1,1))
plot(fit, age=seq(20,80,length=100), sex=NA, conf.int=FALSE)
                         # Plot relationship between age and log
```

```
                          # odds, separate curve for each sex,
                          # no C.I.
z <- plot(fit, age=NA, sex=NA, label.curves=FALSE)
                          # use label.curves=list(keys=c('a','b'))'
                          # to use 1-letter abbreviations
datadensity(z, age, sex) # rug plots (1-dimensional scatterplots)
                          # on each treatment curve, with treatment-
                          # specific density of age
plot(fit, age=seq(20,80,length=100), sex='male')  # works if datadist not used
plot(fit, age=NA, cholesterol=NA)# 3-dimensional perspective plot for age,
                          # cholesterol, and log odds using default
                          # ranges for both variables
boundaries <- perimeter(age, cholesterol, lowess=TRUE)
plot(age, cholesterol)   # show bivariate data density
lines(boundaries)        # and perimeter that will be used for 3-D plot
z <- plot(fit, age=NA, cholesterol=NA, perim=boundaries, method='image')
                          # draws image() plot
                          # don't show estimates where data are sparse
                          # doesn't make sense here since vars don't interact
if(!.R.)Legend(z, fun=plogis, at=qlogis(c(.01,.05,.1,.2,.3,.4,.5)),
               zlab='Probability')   # gray scale or color legend for prob.
plot(fit, age=NA, fun=function(x) 1/(1+exp(-x)) , # or fun=plogis
     ylab="Prob", conf.int=.9)    # Plot estimated probabilities instead of
                                  # log odds

# 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
plot(fit, age=NA, ref.zero=TRUE, fun=exp, ylab='Age=x:Age=30 Odds Ratio')
abline(h=1, lty=2, col=2); abline(v=30, lty=2, col=2)

# Make two curves, and plot the predicted curves as two trellis panels
w <- plot(fit, age=NA, sex=NA)   # Would be nice if a pl=FALSE option was avail.
z <- data.frame(w$x.xbeta)     # Makes variable names legal
if(.R.) library(lattice)
xyplot(log.odds ~ age | sex, data=z, type='l')
# To add confidence bands we need to use the Hmisc xYplot function in
# place of xyplot
xYplot(Cbind(log.odds,lower,upper) ~ age | sex, data=z,
       method='bands', type='l')
# If non-displayed variables were in the model, add a subtitle to show
# their settings using title(sub=paste('Adjusted to',w$adjust),adj=0)
# See predict.Design for an example using predict and xYplot without plot()



# 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)
Srv <- Surv(t,e)

# Fit log-normal survival model and plot median survival time vs. age
f <- psm(Surv(t, e) ~ rcs(age), dist=if(.R.)'lognormal' else 'gaussian')
med <- Quantile(f)       # Creates function to compute quantiles
                         # (median by default)
plot(f, age=NA, fun=function(x)med(lp=x), ylab="Median Survival Time")
# Note: This works because med() expects the linear predictor (X*beta)
#       as an argument.  Would not work if use
#       plot(..., 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
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(f, x1=NA, fun=smean, ylab='Predicted Mean on y-scale')

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.  scat1d is used to indicate
# the income-interval-specific data density for college.  For
# this purpose show the distribution of percent in college for
# those having an income level within +/- the half-width of
# the income interval.  scat1d shows the rug plot superimposed
# on the estimated curve.  Data are county-level percents.
# This can't be done automatically using datadensity on the object
# returned by plot.Design, as the variable representing different
```

```
# curves (income) is a continuous variable.

incomes <- seq(22900, 32800, length=4)
# equally spaced to outer quintiles
pl <- plot(f, college=NA, income=incomes,
           conf.int=FALSE, xlim=c(0,35), ylim=c(30,55),
           lty=1, lwd=c(.25,1.5,3.5,6), col=c(1,1,2,2))
graph.points <- pl$x.xbeta
for(i in 1:4) {
  college.in.income.group <- college[abs(income-incomes[i]) < 1650]
  this.income <- graph.points[,'income']==incomes[i]
  scat1d(college.in.income.group,
         curve=list(x=graph.points[this.income,'college'],
           y=graph.points[this.income,'democrat']))
}

# Instead of showing a rug plot on each curve, erase end portions
# of each curve where there are fewer than 10 counties having
# % 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(f, college=NA, income=incomes,
     conf.int=FALSE, xlim=c(0,35), ylim=c(30,55),
     lty=1, lwd=c(.25,1.5,3.5,6), col=c(1,1,2,2),
     perim=show.pts)
## End(Not run)
```

`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, ...)
```

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.
...	other arguments passed to plot and lines

Side Effects

plots

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
⟨f.harrell@vanderbilt.edu⟩

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, cumcategory, chiSquare.

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)
```

---

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':
Varcov(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
f.harrell@vanderbilt.edu

See Also

psm, summary.Design, print.pphsm

Examples

```
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 prints a matrix of asterisks showing which factors were selected at each repetition, along with a frequency distribution of the number of factors retained across resamples.

Usage

```
predab.resample(fit.orig, fit, measure,
                method=c("boot","crossvalidation",".632","randomization"),
                bw=FALSE, B=50, pr=FALSE,
                rule="aic", type="residual", sls=.05, aics=0,
                strata=FALSE, tol=1e-12, non.slopes.in.x=TRUE, kint=1,
                cluster, subset, group=NULL, ...)
```

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.
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. For method="crossvalidation", this is also the number of groups the original sample is split into.

pr — TRUE to print results for each sample. Default is FALSE.

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.

strata — set to TRUE if fit.orig has an x element that contains a "strata" attribute which is a vector that should be sampled the same way as the observations in x and y

tol — Tolerance for singularity checking. Is passed to fit and 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.

... — 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 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

Side Effects

prints a summary of the results

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

References

Efron B, Tibshirani R (1997). Improvements on cross-validation: The .632+ bootstrap method. JASA 92:548–560.

See Also

`Design`, `validate`, `fastbw`, `lrm`, `ols`, `cph`, `bootcov`, `naresid`

Examples

```
# See the code for validate.ols for an example of the use of
# predab.resample
```

predict.Design *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.

Usage

```
## S3 method for class 'bj':
predict(object, newdata,
        type=c("lp", "x", "data.frame",
                 "terms", "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE, conf.type=c('mean','individual'),
        incl.non.slopes,
        non.slopes, kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=TRUE, ...) # for bj

## S3 method for class 'cph':
predict(object, newdata,
        type=c("lp", "x",
                 "data.frame", "terms", "adjto", "adjto.data.frame",
                 "model.frame"),
        se.fit=FALSE, conf.int=FALSE, conf.type=c('mean','individual'),
        incl.non.slopes=NULL,
        non.slopes=NULL, kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=TRUE, ...) # cph

## S3 method for class 'glmD':
predict(object, newdata,
        type= c("lp", "x", "data.frame",
                 "terms", "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE, conf.type=c('mean','individual'),
        incl.non.slopes,
        non.slopes, kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=TRUE, ...) # glmD

## S3 method for class 'glsD':
predict(object, newdata,
        type=c("lp", "x", "data.frame",
                 "terms", "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE, conf.type=c('mean','individual'),
```

```
        incl.non.slopes,
        non.slopes, kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=TRUE, ...) # glsD

## S3 method for class 'ols':
predict(object, newdata,
        type=c("lp", "x", "data.frame",
                 "terms", "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE, conf.type=c('mean','individual'),
        incl.non.slopes,
        non.slopes, kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=TRUE, ...) # ols

## S3 method for class 'psm':
predict(object, newdata,
        type=c("lp", "x", "data.frame",
                 "terms", "adjto", "adjto.data.frame", "model.frame"),
        se.fit=FALSE, conf.int=FALSE, conf.type=c('mean','individual'),
        incl.non.slopes,
        non.slopes, kint=1, na.action=na.keep, expand.na=TRUE,
        center.terms=TRUE, ...) # psm
```

Arguments

`object`	a fit object with a `Design` 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,

"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. 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.

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.

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".

incl.non.slopes
Default is TRUE if non.slopes or kint is specified, the model has a scale parameter (e.g., a parametric survival model), or type!="x". Otherwise the default is FALSE. Set to TRUE to use an intercept in the prediction if the model has any intercepts (except for type="terms" which doesn't need intercepts). Set to FALSE to get predicted $X\beta$ ignoring intercepts.

non.slopes For models such as the ordinal logistic models containing more than one intercept, this specifies dummy variable values to pick off intercept(s) to use in computing predictions. For example, if there are 3 intercepts, use non.slopes=c(0,1,0) to use the second. Default is c(1,0,...,0). You may alternatively specify kint.

kint a single integer specifying the number of the intercept to use in multiple-intercept models

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 the mean from columns of the design matrix before computing terms with type="terms".

... ignored

Details

datadist and options(datadist=) should be run before predict.Design 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
f.harrell@vanderbilt.edu

See Also

plot.Design, summary.Design, Design, Design.trans, predict.lrm, residuals.cph, naresid, datadist, gendata, Function.Design, reShape, xYplot, contrast.Design

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))
# 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
#   logit <- plot(f, age=ag, ...)$x.xbeta[,2]
#which allows evaluation of the shape for any level of interacting
#factors.  When age does not interact with anything, the result from
#predict(f, ..., 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

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$coef
cov   <- fit$var
```

```
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 <- 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.Design

## 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.Design) 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)
```

predict.lrm *Predicted Values for Binary and Ordinal Logistic Models*

Description

Computes a variety of types of predicted values for fits from `lrm`, either from the original dataset or for new observations. The `Mean.lrm` function produces an S function to compute the predicted mean of a numeric ordered response variable given the linear predictor, which is assumed to use the first intercept in its computation.

Usage

```
## S3 method for class 'lrm':
predict(object, ..., type=c("lp", "fitted", "fitted.ind", "mean", "x",
            "data.frame", "terms", "adjto","adjto.data.frame",
            "model.frame"), se.fit=FALSE, codes=FALSE)

## S3 method for class 'lrm':
Mean(object, codes=FALSE, ...)
```

Arguments

`object`	a object created by `lrm`
`...`	arguments passed to `predict.Design`, such as `kint` and `newdata` (which is used if you are predicting `out of data`). See `predict.Design` to see how NAs are handled.
`type`	See `predict.Design` for `"x", "data.frame", "terms", "adjto", "adjto.data.frame"` and `"model.frame"`. `type="lp"` is used to get linear predictors (always using the first intercept). `type="fitted"` is used to get all the probabilities $Y \geq j$. `type="fitted.ind"` gets all the individual probabilities $Y = j$. 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"` or `Mean.lrm` uses 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` the result is an S function.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

References

Hannah M, Quigley P: Presentation of ordinal regression analysis on the original scale. Biometrics 52:771–5; 1996.

See Also

lrm, predict.Design, naresid, contrast.Design

Examples

```
# See help for predict.Design 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)
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 plot.Design or nomogram to
# get predicted means instead of log odds or probabilities
# Don't use non.slopes argument to plot.Design for this
dd <- datadist(x1,x2); options(datadist='dd')
m
plot(f, x1=NA, fun=m, ylab='Predicted Mean')
```

---

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, VIFs.

Usage

```
## S3 method for class 'cph':
print(x, long=FALSE, digits=3, conf.int=FALSE, table=TRUE, ...)
```

Arguments

x	fit object
long	set to TRUE to print the centering constant
digits	number of significant digits to print
conf.int	set to e.g. .95 to print 0.95 confidence intervals on simple hazard ratios
table	set tp FALSE to suppress event frequency statistics
...	arguments passed to print.cphfit : coef, scale

See Also

print.coxph

---

print.cph.fit *Print cph.fit*

Description

Formatted printing of an object of class cph.fit created by cph.fit (which is usually called by cph). Most of the logic for print.cph.fit came from Therneau's print.coxreg.

Usage

```
## S3 method for class 'cph.fit':
print(x, table=TRUE, coef=TRUE, conf.int=FALSE, scale=1, digits=NULL,...)
```

Arguments

x	object created by cph.fit
table	print table of frequencies of events, by strata (if any)
coef	print coefficient estimates, standard errors, and z-statistics
conf.int	set to e.g. .95 to print 0.95 confidence intervals. Default is FALSE to suppress confidence intervals
scale	constant by which to multiply coefficients and standard errors if printing confidence intervals.
digits	number of significant digits to print
...	ignored

See Also

print.coxph

print.lrm *Print lrm*

Description

Formatted printing of an object of class `lrm`

Usage

```
## S3 method for class 'lrm':
print(x, digits=4, strata.coefs=FALSE, ...)
```

Arguments

`x`	fit object
`digits`	number of significant digits to use
`strata.coefs`	set to `TRUE` to print the (experimental) strata coefficients
`...`	ignored

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.

Usage

```
## S3 method for class 'ols':
print(x, digits=4, long=FALSE, ...)
```

Arguments

`x`	fit object
`digits`	number of significant digits to print
`long`	set to `TRUE` to print the correlation matrix of parameter estimates
`...`	other parameters to pass to `print` or `format`

See Also

`ols`, `print.lm`, `summary.lm`

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 Design 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 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 survfit to get Kaplan-Meier estimates, with optional stratification (automatically grouping a continuous variable into quantiles) and then through survplot.survfit to plot them. Then lines is invoked to show the theoretical curve. Other types of residuals are computed by residuals using residuals.survreg.

Older versions of survreg used by psm (e.g., on S-Plus 2000) had the following additional arguments method, link, parms, fixed. See survreg on such systems for details. psm passes those arguments to survreg.

Usage

```
psm(formula=formula(data),
    data=if (.R.) parent.frame() else sys.parent(), weights,
    subset, na.action=na.delete, dist="weibull",
    init=NULL, scale=0,
    control=if(!.R.) survReg.control() else survreg.control(),
    parms=NULL,
    model=FALSE, x=FALSE, y=TRUE, time.inc, ...)
# dist=c("extreme", "logistic", "gaussian", "exponential",
#        "rayleigh", "t")      for S-Plus before 5.0
# dist=c("extreme", "logistic", "gaussian", "weibull",
#        "exponential", "rayleigh", "lognormal",
#        "loglogistic" "t")    for R, S-Plus 5,6
# Older versions had arguments method, link, parms, fixed

## S3 method for class 'psm':
print(x, correlation=FALSE, ...)

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="censored.normalized", ...)

## 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 (survReg for S-Plus 5. or 6.). fixed is used for S-Plus before 5., parms is used for S-Plus 5, 6, and R. See cph for na.action.
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
...	other arguments to fitting routines, or to pass to survplot from survplot.residuals.psm.censored.normalized. Ignored 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 (survReg for S-Plus 6).
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 Design 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
f.harrell@vanderbilt.edu

See Also

Design, survreg, survReg, residuals.survreg, survreg.object, survreg.distributions, pphsm, survplot, survest, Surv, na.delete, na.detail.response, datadist, latex.psm

Examples

```
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=if(.R.)'lognormal' else 'gaussian')
# Log-normal model is a bad fit for proportional hazards data

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, dxy=TRUE, B=10)      # ordinarily use B=150 or more
plot(f, age=NA, sex=NA)       # needs datadist since no explicit age, hosp.
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=if(.R.)expression(X*hat(beta)) else 'X*Beta',
     ylab=if(.R.)expression(S(6,X*hat(beta))) else 'S(6|X*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=if(.R.)"weibull" else "extreme")
f
pphsm(f)          # print proportional hazards form
g <- survest(f)
plot(g$time, g$surv, xlab='Time', type='l',
     ylab=if(.R.)expression(S(t)) else 'S(t)')

f <- psm(Surv(d.time,death) ~ age,
         dist=if(.R.)"loglogistic" else "logistic", y=TRUE)
r <- resid(f, 'cens') # note abbreviation
survplot(survfit(r), conf='none')
                      # plot Kaplan-Meier estimate of
                      # survival function of standardized residuals
survplot(survfit(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)
```

`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"), collapse, weighted, ...)
```

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.
`collapse`	Vector indicating which rows to collapse(sum) over. In time-dependent models more than one row data can pertain to a single individual. If there were 4 individuals represented by 3, 1, 2 and 4 rows of data respectively, then `collapse=c(1,1,1, 2, 3,3, 4,4,4,4)` could be used to obtain per subject rather than per observation residuals.
`weighted`	ignored; only accepts `FALSE`
`...`	unused

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

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.lrm` *Residuals from a Logistic Regression Model 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 ($Y - P$), 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. 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. 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("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)
```

Arguments

object	object created by lrm
...	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 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`.

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 last example.

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
f.harrell@vanderbilt.edu

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.

See Also

`lrm`, `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
  if(.R.) {
    plot(xx, r[,i], xlab=c('x1','x2')[i])
    lines(lowess(xx,r[,i]))
  } else {
    g <- loess(r[,i] ~ xx)
    plot(g, coverage=0.95, confidence=7)
    points(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)
# 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



# 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)
  if(!.R.) dim(p) <- dim(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))

# Get data used in Hosmer et al. paper and reproduce their calculations
if(FALSE && .R.) {
v <- Cs(id, low, age, lwt, race, smoke, ptl, ht, ui, ftv, bwt)
d <- read.table("http://www-unix.oit.umass.edu/~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
}
```

`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"), ...)
```

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.
`...`	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
f.harrell@vanderbilt.edu

See Also

`lm.influence`, `ols`, `which.influence`, `naresid`

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)
```

---

rm.impute *Imputation of Repeated Measures*

Description

NOTE: This function is under development and is not correct at present. Uses the method of Lavori, Dawson, and Shera (1995) to analyze uniformly (over subjects) collected repeated measurement data subject to non-random dropout. Separately for each imputation and for each time period, a binary logistic model is developed (using the Design `lrm` function) to predict the probability that each subject remains in the study at that time period. Predictors for the first time period are those listed in the `pformula` formula. These are assumed to be baseline variables that are never missing. For later time periods, predictors include the baseline predictors plus the matrix of response (`y`) values for all earlier periods. These "previous responses" will have missing values imputed from the earlier steps.

Missing responses for time period i are imputed, for one of the `n.impute` multiple imputations, as follows. The period i fitted propensity model described above is evaluated to obtain the predicted probability that each subject remained in the study until at least period i. The estimated propensity is divided into `g` quantile groups. If for period i within a propensity quantile group there are `a` subjects still in the study and `b` subjects who have dropped out, Rubin's approximate Bayesian bootstrap is used to estimate the predictive distribution of the response values for the `b` dropouts, given that the propensity for remaining in the study is approximately constant for all subjects (dropouts and non-dropouts) in the group. A sample of size `a` is selected with replacement from the `a` subjects still in the study from the propensity group. Then a sample of size `b` with replacement is selected from this sample of size `a`. These `b` responses are used to fill-in the responses for the `b` dropouts in the quantile group for the current imputation and current time period.

If the right-hand-side of a formula is specified for a univariate response summary (which may be the last response, mean, or area under the time-response curve), `rm.impute` goes on to fit `rformula` to this response summary for each of the multiple imputations using a fitting function `fitter`. After all `n.impute` imputations have been done, the average "apparent" covariance matrix and the between-imputation covariance matrix are computed to derive Rubin's multiple-imputation-corrected covariance matrix for the average of `n.impute` sets of regression coefficients. See `fit.mult.impute` for more details.

The response variable `y` may be an array to handle multiple responses at each time period. This array has number of rows equal to the number of subjects, number of columns equal to the number of periods, and number of "pages" equal to the number of different response measurements. A utility function `pbind` is supplied for creating such arrays from a series of matrices. When multiple

responses are present, all responses are used in the current propensity model, and the `which`, `nk`, `rinteraction`, and `rint.with` arguments will apply equally to all responses.

Usage

```
rm.impute(pformula, y, last,
          rformula, fitter=ols, which=c("last", "mean", "auc"),
          data=sys.parent(1), n.impute=10, g=5,
          nk=0, rinteraction, rint.with=c('all','recent'),
          pr=FALSE, pra=FALSE, npr,
          keep.prop=FALSE, keep.pfits=FALSE)

pbind(...)
```

Arguments

`pformula`	right-hand-side propensity formula, e.g., treatment + x1 + x2 + x3*x4. This formula (as well as `rformula` if `fitter` is one of the Design library fitting functions) can contain any of the Design library's transformation functions such as `rcs`, `pol`, etc.
`y`	matrix of responses over time periods. To use `which="auc"`, column names of `y` must contain numeric measurement times.
`last`	an integer vector whose value for the jth subject is the last period before the subject dropped out. A subject who never had a follow-up response measured will have `last=0`.
`...`	a series of matrices to "page bind" into an array. New names may be supplied using the notation `pbind(newname1=y1,newname2=y2)`. The `dimnames` of the first argument (which will be converted to a matrix if it is a vector, for the unusual one-period case) will be used as the first two `dimnames` of the resulting array, and the names of the matrices will form the third vector of `dimnames`.
`rformula`	right-hand-side response formula, e.g., x1 + pol(x2) + treatment. If omitted, `rm.impute` will return only the multiple response imputations.
`fitter`	any S-Plus or Design library fitting function for a univariate response summary. The default is `ols`. If there are multiple response variables at each time period and you want to use a different fitter for different response variables, specify a list of `nr` fitting functions as this argument, where `nr` is the number of response variables.
`which`	which response summary is used if `rformula` is given. The default is the last column of the response matrix.
`data`	usually a data frame, if the variables in `pformula` and `rformula` are not already available via `attach()`
`n.impute`	number of imputations. The more missing data, the higher `n.impute` should be.
`g`	number of propensity quantile groups
`nk`	number of knots to use in expanding each previous response into a restricted cubic spline in the propensity model. Default is zero (assume linearity).

rinteraction	a character vector specifying the names of baseline variables that should be interacted with each response in the propensity model. Default is no such interactions.
rint.with	set to "recent" to allow the variables in rinteraction to only interact with the response for the most recent time period, and not with the most recent and all previous responses (the default)
pr	set to TRUE to print each logistic propensity model fit.
pra	if pr=TRUE, you can also set pra=TRUE to print the Design anova() results for each propensity model fit.
npr	if pr=TRUE, printing will be done for the first npr imputations
keep.prop	set to TRUE to store the array propensity in the returned list. The dimensions for propensity are the same as Y.
keep.pfits	set to TRUE to store all propensity model fits from lrm in the result returned by rm.impute

Details

The algorithm used here will not correct for non-random dropout due to variables that are not included in the propensity model. A worst-case would be having dropouts at period i due to unmeasured responses at period i.

Ironically, there must be a sufficient number of dropouts for the propensity score method to work, as the propensity models must have adequate numbers of dropouts and non-dropouts at each time period.

Value

a list with elements Y and optionally fit (if rformula is given) and propensity (if keep.prop=TRUE). Y and propensity are arrays whose last dimension corresponds to the multiple imputations and whose first two dimensions correspond to y. Y is the multiply-imputed response array and fit is the imputation-corrected fit object. Note: Aside from the regression coefficient vector and covariance matrix, this fit object will have parameters from the fit of the response summary for the last imputation. If keep.pfits=TRUE, the returned list will also have an array of propensity fit objects (lrm objects) for all response periods and imputations. If there is more than one response variable at each time period, fit will be a list of nr fit objects for nr response variables.

Side Effects

prints, and creates variables such as y.1, y.2, ... and in.period.i in the session database (frame 0)

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University School of Medicine
f.harrell@vanderbilt.edu
Much valuable input was received from Chris Barker (Roche Pharmaceuticals) and Phil Lavori (Stanford University).

References

Lavori PW, Dawson R, Shera, D: A multiple imputation strategy for clinical trials with truncation of patient data. Stat in Med 14:1913–1925, 1995.

Rubin D, Shenker N: Multiple imputation in health-care data bases: An overview and some applications. Stat in Med 10:585–598, 1991.

Engels JM, Diehr P: Imputation of missing longitudinal data: a comparison of methods. J Clin Epi 56:968–976, 2003.

See Also

transcan, fit.mult.impute, lrm, rm.boot, reShape

Examples

```
## Not run:
# Generate multiple imputes of the response matrix for later use
Y <- rm.impute(~treatment + pol(age,2)*sex, responses,
               last=lastvisit, data=mydata)$Y
# Do some analysis for each imputation
fits <- vector('list',10)
for(i in 1:10) {
  y <- Y[,,i]
  fits[[i]] <- my.analysis(X,y)
}

# Function to generate a 4-variate equal correlation pattern response
# with missing-at-random responses; missingness is a function of x and
# previous responses.
#
# pna is a function that computes the probability that a subject
# drops out at the current visit.  For visit 1 pna is a function
# of treatment and baseline covariable x.  For visits > 1 pna is
# a function of the matrix of responses for all previous visits.
#
# If second=TRUE we also generate a second response variable having
# NAs in the same positions as this first one.  y2 is generated
# so that its NAs are completely unrelated to any y2 values if
# y2B.effect=0, as the pna function is only given the first
# response variable.
# y2 is N(0,1) for treat='A' and N(y2.treat.effect,1) for treat='B'.

testdf <- function(n=1500, seed=7, pna, second=FALSE, y2.treat.effect=0) {

set.seed(seed)
treat <- sample(c('A','B'),n,TRUE)
x <- runif(n)
nt <- 4

mvrnorm <- function(n, p = 1, u = rep(0, p), S = diag(p)) {
  Z <- matrix(rnorm(n * p), p, n)
  t(u + t(chol(S)) %*% Z)
```

```
}

# Generate multivariate normal errors for n subjects at nt times
# Assume equal correlations of rho=.5, independent subjects

rho   <- .5
y <- mvrnorm(n, p=nt, S=diag(rep(1-rho,nt))+rho)

y[treat=='B',] <- y[treat=='B',] + 1

cat('\n\nTreatment-specific means for last period in response variable 1 before generating N
print(tapply(y[,4], treat, mean, na.rm=TRUE))

y[runif(n) < pna(treat, x), 1] <- NA
y[is.na(y[,1]) | runif(n) < pna(treat, x, y[,1]),   2] <- NA
y[is.na(y[,2]) | runif(n) < pna(treat, x, y[,1:2]), 3] <- NA
y[is.na(y[,3]) | runif(n) < pna(treat, x, y[,1:3]), 4] <- NA

last <- rep(4, n)
last[is.na(y[,4])] <- 3
last[is.na(y[,3])] <- 2
last[is.na(y[,2])] <- 1
last[is.na(y[,1])] <- 0

cat('\nNumber of NAs for each time period:\n')
print(apply(y, 2, function(x)sum(is.na(x))))
cat('\n\nTreatment-specific means for last period in response variable 1 after excluding NAs
print(tapply(y[,4], treat, mean, na.rm=TRUE))
cat('\n\nNaive complete-case analysis:\n\n')
prn(ols(y[,4] ~ pol(x,2) + treat))

if(second) {
  y2 <- matrix(rnorm(n*4),ncol=4)
  y2[treat=='B',] <- y2[treat=='B',] + y2.treat.effect
  cat('\n\nTreatment-specific means for last period in response variable 2 before generating
  print(tapply(y2[,4], treat, mean, na.rm=TRUE))

  y2[is.na(y[,1]),1] <- NA
  y2[is.na(y[,2]),2] <- NA
  y2[is.na(y[,3]),3] <- NA
  y2[is.na(y[,4]),4] <- NA
  cat('\n\nTreatment-specific means for last period in response variable 2 after excluding N
  print(tapply(y2[,4], treat, mean, na.rm=TRUE))

  y <- pbind(y1=y, y2=y2)
}

list(x=x, treat=treat, y=y, last=last)
}

pna <- function(treat, x, yprev) {
# In this model for the probability of dropout just before the
# current visit, the probability does not depend on the baseline
```

```
# covariable x.  For treat='B' the probability of dropout is a
# constant 0.1.  For treat='A' it is a curtailed quadratic
# function of the previous visit's response.
#
# If no previous responses available, we are at first follow-up visit

if(missing(yprev)) 0 else {
  if(is.matrix(yprev)) yprev <- yprev[,ncol(yprev)]
  ifelse(treat=='B', .1,
         pmax(0, pmin(1, .124 +.0835*yprev + .020868*yprev^2)))
  }
}

df <- testdf(pna = pna, second=TRUE)

g <- rm.impute(~ pol(x,2) + treat, df$y, last=df$last,
               rformula=~ pol(x,2) + treat,
               n.impute=10, g=4, nk=3,
               rinteraction='treat', rint.with='all',
               pr=TRUE, pra=TRUE, data=df, keep.prop=TRUE, keep.pfits=TRUE)
# Base propensity model is in.study ~ pol(x,2) + treat
# for visits 2,3,4, filled-in y's from previous visits will also be
# used as predictors, and these interact with treat.
# Restricted cubic spline with 3 knots is assumed for the propensity models
# To fit the multiply-imputed last (4th) response an additive model
# in quadratic x and treat is used

g$fit[[1]]       # shows response fit for first response variable
                 # (y1), with variances adj. for imputation
page(g$Y)        # show all 10 imputations for both responses x 4 periods

# Check for the first imputation how well propensity matching achieved
# balance in baseline and period 3 filled-in responses for
# dropouts and non-dropouts.  For continuous variables show ECDFs
# using the Hmisc ecdf function, for first 4 imputations.  Do this
# with and without stratifying on quintiles of propensity, and also
# show the estimated 3rd period response  vs. propensity stratified
# by dropout status.  Use only first response (y1) for all of this.

for(imp in 1:4) {
  y3     <- g$Y[,3,1,imp]
  prop3  <- g$propensity[,3,imp]
  prop3g <- cut2(prop3,g=5)
  ti <- paste('Imputation',imp)
  print(ecdf(~ y3, groups=df$last >= 3, subset=unclass(prop3g)<5))
  title(ti)
  print(ecdf(~ y3 | prop3g, groups=df$last >= 3,
             subset=unclass(prop3g)<5))
  # Not enough dropouts in highest quintile of propensity completing
  # visit 3
  title(ti)
  plsmo(prop3, y3, group=df$last >= 3, datadensity=TRUE, col=1:2)
  title(ti)
```

```
}

# Examine propensity fit for sixth imputation, 4th response
f <- g$pfits[4,6][[1]]
dfr <- as.data.frame(df)
# Edit names of dfr so that responses called y.1, y.2, etc.
# For this example, these are already OK
dd <- datadist(dfr)
options(datadist='dd')
# datadist makes plot below work without specifying variable settings
plot(f, y.3=NA, treat=NA, conf.int=FALSE)

# Analyze multiple response variables.  Both systolic.bp and
# diastolic.bp are matrices (columns = time periods)

f <- rm.impute(~treatment + pol(age,2)*sex,
               pbind(systolic.bp, diastolic.bp),
               last=lastvisit, data=mydata)

# To deal with a continuous and a binary endpoint you can specify
# pbind(sysbolic.bp, stroke), fitter=list(ols, lrm)
## 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 Design() 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.

Warnings

Adjusted ols fits do not have the corrected standard errors printed with print.ols. Use sqrt(diag(adjfit$var)) to get this, where adjfit is the result of robcov.

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

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

Examples

```
# 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(1)
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(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 <- 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)
```

`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
f.harrell@vanderbilt.edu, 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.

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

# 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
```

---

specs.Design *Design 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 'Design':
specs(fit, long=FALSE, ...)

## S3 method for class 'specs.Design':
print(x, ...)
```

Arguments

fit	a fit object created with the Design 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
f.harrell@vanderbilt.edu

See Also

Design, Design.trans, latex.Design, 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 <- glmD(sick ~ rcs(blood.pressure,5), family=binomial)
specs(g,long=TRUE)
options(datadist=NULL)
```

---

summary.Design *Summary of Effects in Model*

Description

summary.Design 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, inter-quartile 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.Design prints the results, latex.summary.Design typesets the results, and plot.summary.Design 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: .7, .8, .9, .95, and .99, using levels of gray scale (colors for Windows).

Usage

```
## S3 method for class 'Design':
summary(object, ..., est.all=TRUE, antilog,
conf.int=.95, abbrev=FALSE, vnames=c("names","labels"))

## S3 method for class 'summary.Design':
print(x, ...)

## S3 method for class 'summary.Design':
latex(object, title, ...)

## S3 method for class 'summary.Design':
plot(x, at, log=FALSE,
    q=c(0.7, 0.8, 0.9, 0.95, 0.99), xlim, nbar, cex=1, nint=10,
    cex.c=.5, cex.t=1, clip=c(-1e30,1e30), main, ...)
```

Arguments

object	a Design 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=NA, sex=NA 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, or other optional arguments passed to confbar. The most important of these is q, the vector of confidence levels, and col, which is a vector corresponding to q specifying the colors for the regions of the bars. q defaults to c(.7,.8,.9,.95,.99) and col to c(1,.8,.5,.2,.065) for UNIX, so that lower confidence levels (inner regions of bars) corresponding with darker shades. Specify for example col=1:5 to use actual colors. For Windows, the default is col=c(1,4,3,2,5), which by default represents black, blue, green, red, yellow.
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.
x	result of summary
title	title to pass to latex. Default is name of fit object passed to summary prefixed with "summary".
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.c	cex parameter for confbar, for quantile labels.
cex.t	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".

Value

For summary.Design, a matrix of class summary.Design 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.Design returns an object of class c("latex","file"). It requires the latex function in Hmisc.

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

See Also

datadist, Design, Design.trans, Design.Misc, confbar, pretty, contrast.Design

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 screen
## 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)
```

`summary.survfit` *Design version of survival Package summary.survfit*

Description

This is a modified version of the survival packages's `summary.survfit` function. It returns a list containing the survival curve, confidence limits, and other information.

Usage

```
## S3 method for class 'survfit':
summary(object, times, censored = FALSE, scale = 1, ...)
```

Arguments

`object`	result of `survfit`
`times`	
`censored`	
`scale`	
`...`	see `summary.survfit`

Value

a list with components `time,surv,n.risk,n.event,std.err,conf.int,lower,upper,strata,call,na`

See Also

`summary.survfit`

Examples

`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(-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-log", the default, is the most natural basis.
se.fit	set to TRUE to get standard errors of log-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
f.harrell@vanderbilt.edu

See Also

cph, survfit.cph, survfit.coxph, predict.Design, 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
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))
options(datadist=NULL)
```

`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 i^{th} subject's survival probability at the i^{th} 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 "survfit" so the survival curve can be plotted with survplot.survfit. 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
f.harrell@vanderbilt.edu

See Also

psm, survreg, Design, survfit, predict.Design, survplot, survreg.distributions

Examples

```
# Simulate data from a proportional hazards population model
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` *Modified Version of survival Package survfit Function*

Description

This modification of `survfit` keeps attributes of the `Surv` object, users `Hmisc`'s `interaction()` to form strata labels, and uses a default confidence interval basis of log-log.

Usage

```
survfit(formula, data, weights, subset, na.action = na.delete, conf.type = c("log-l
```

Arguments

`formula`	
`data`	
`weights`	
`subset`	
`na.action`	
`conf.type`	
`...`	see `survfit`

Value

see `survfit`

See Also

`survfit`

Examples

```
## Not run:
#fit a Kaplan-Meier and print the results
data(aml)
survfit(Surv(time, status) ~ x, data=aml)
## End(Not run)
```

---

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. Unlike survfit.coxph, the default basis for confidence intervals is "log-log".

Usage

```
survfit.cph(object, newdata, se.fit=TRUE, conf.int=0.95,
            individual=FALSE, type, vartype,
            conf.type=c("log-log", "log", "plain", "none"))
```

Arguments

object	a fit object from cph or coxph see survfit.coxph
newdata	
se.fit	
conf.int	
individual	
type	
vartype	
conf.type	see survfit

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. 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.

Usage

```
survplot(fit, ...)
## S3 method for class 'Design':
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("bars","bands"),
         add=FALSE, label.curves=TRUE,
         abbrev.label=FALSE, lty, lwd=par("lwd"), col=1,
         adj.subtitle, loglog=FALSE, fun,
         n.risk=FALSE, logt=FALSE, dots=FALSE, dotsize=.003,
         grid=FALSE, srt.n.risk=0, sep.n.risk=0.056, adj.n.risk=1,
         y.n.risk, cex.n.risk=.6, pr=FALSE)
## S3 method for class 'survfit':
survplot(fit, xlim,
         ylim, xlab, ylab, time.inc,
         conf=c("bars","bands","none"), add=FALSE,
         label.curves=TRUE, abbrev.label=FALSE,
         lty,lwd=par('lwd'),col=1,
         loglog=FALSE,fun,n.risk=FALSE,logt=FALSE,
         dots=FALSE,dotsize=.003,
         grid=FALSE,
         srt.n.risk=0,sep.n.risk=.056,adj.n.risk=1,
         y.n.risk,cex.n.risk=.6, pr=FALSE, ...)
```

Arguments

`fit`	result of fit (`cph`, `psm`, `survfit`, `survest.psm`)
`...`	list of factors with names used in model. For fits from `survfit`, 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 NA (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). NA may be specified only for the first factor. 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 survfit 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.

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".

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.

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 survfit, this argument is also ignored, since it is specified as an argument to survfit.

conf.type — specifies the basis for confidence limits. If estimates stored with surv=TRUE are being used, always uses "log-log", the default. This argument is ignored for fits from survfit.

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 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 survfit, which must have previously specified confidence interval specifications.

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" for bands using standard errors at each failure time. For survfit objects only, conf may also be "none", indicating that confidence interval information stored with the survfit result should be ignored.
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(metho 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
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.
adj.subtitle	set to FALSE to suppress plotting subtitle with levels of adjustment factors not plotted. Defaults to TRUE if there are 4 or fewer adjustment factors. This argument is ignored for survfit.
loglog	set to TRUE to plot log(-log Survival) instead of Survival
fun	specifies any function to translate estimates and confidence limits before plotting
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 survfit. 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.
cex.n.risk	character size for number of subjects at risk (when n.risk is TRUE)
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 FALSE. Set to a color shading to plot faint lines. Set to 1 to plot solid lines. Default is .05 if TRUE.
pr	set to TRUE to print survival curve coordinates used in the plots

Details

survplot will not work for Cox models with time-dependent covariables. Use survest or survfit for that purpose.

Use ps.slide, win.slide, gs.slide to set up nice defaults for plotting. These also 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 survfit, 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).

See Also

datadist, Design, cph, psm, survest, predict.Design, plot.Design, units, errbar, survfit, survreg.distributions, labcurve, mgp.axis.labels, par, ps.slide

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
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)

#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), surv=TRUE)
#or f <- psm(S ~ pol(age,2)*sex)

survplot(f, sex=NA, n.risk=TRUE)           #Adjust age to median
survplot(f, sex=NA, 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=NA, label.curves=list(keys=c(2,0), point.inc=2))
                                           #Identify curves with symbols

survplot(f, sex=NA, 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,seq(0,1,by=.2)), sex="male")

#Plot survival Kaplan-Meier survival estimates for males
f <- survfit(S, subset=sex=="male")
survplot(f)

#Plot survival for both sexes
f <- survfit(S ~ sex)
survplot(f)
#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)")

options(datadist=NULL)
```

`val.prob` *Validate Predicted Probabilities*

Description

The `val.prob` and `val.surv` functions are useful for validating predicted probabilities against binary events and predicted survival probabilities against right-censored failure times, respectively. First `val.prob` is described.

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`, and E_{max}=maximum absolute difference in predicted and calibrated probabilities. 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.

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`.

`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.

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(0,0.9), riskdist="calibrated", cex=.75, 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, ...)

val.surv(fit, newdata, S, est.surv, censor)

## 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

`p`	predicted probability
`y`	vector of binary outcomes

logit	predicted log odds of outcome. Specify either p or logit.
fit	a fit object created by cph or psm
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. Other defaults are used by val.surv.
ylab	y-axis label, default is "Actual Probability" for val.prob. Other defaults are used by val.surv.
lim	limits for both x and y axes
m	If grouped proportions are desired, average 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	Defaults to "calibrated" to plot the relative frequency distribution of calibrated robabilities after dividing into 101 bins from lim[1] to lim[2]. Set to "predicted" 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) or val.surv
...	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 "$\star$" if either ChiSq2 or B ChiSq is significant at the 0.01 level and " " otherwise.
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
est.surv	a vector of estimated survival probabilities corresponding to times in the first column of S.
censor	a vector of censoring times. Only the censoring times for uncensored observations are used.
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.

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`, `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"`.

Side Effects

plots calibration curve

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

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.

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.

Cox DR, Snell EJ (1968):A general definition of residuals (with discussion). JRSSB 30:248–275.

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.

See Also

`validate.lrm`, `lrm.fit`, `lrm`, `labcurve`, `wtd.rank`, `wtd.loess.noiter`, `scat1d`, `cph`, `psm`

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.

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)
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))

# Survival analysis examples
# Generate failure times from an exponential distribution
set.seed(123)              # so can reproduce results
n <- 2000
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
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
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 `Design` series does resampling validation of a regression model, with or without backward step-down variable deletion.

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,
         pr=FALSE, ...)
```

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.

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".
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 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).

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 rcorr.cens function to be invoked to compute the Somers' D_{xy} rank correlation to be computed at each resample (this takes a bit longer than the likelihood based statistics). The values corresponting 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
f.harrell@vanderbilt.edu

See Also

validate.ols, validate.cph, validate.lrm, validate.tree, predab.resample, fastbw, Design, Design.trans, calibrate

Examples

```
# See examples for validate.cph, validate.lrm, validate.ols
# Example of validating a parametric survival model:

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.

Usage

```
# fit <- cph(formula=Surv(ftime,event) ~ terms, x=TRUE, y=TRUE, ...)
## S3 method for class 'cph':
validate(fit, method="boot", B=40, bw=FALSE, rule="aic", type="residual",
```

```
        sls=.05, aics=0, pr=FALSE, dxy=FALSE, 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, pr=FALSE,
        dxy=FALSE, tol=1e-12, rel.tolerance=1e-5, maxiter=15, ...)
```

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
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".
pr	TRUE to print results of each repetition
tol	
...	see validate or predab.resample
dxy	set to TRUE to validate Somers' D_{xy} using rcorr.cens, which takes longer.
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.

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 $Q = D - U$. 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 corresponting 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 and she wishes to get the more usual correlation using predicted survival time or predicted survival probability, D_{xy} should be 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
f.harrell@vanderbilt.edu

See Also

validate, predab.resample, fastbw, Design, Design.trans, calibrate, rcorr.cens, cph, coxph.fit

Examples

```
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, dxy=TRUE, u=2, B=10)   # normally B=150
# Note u=time.inc
```

validate.lrm *Resampling Validation of a Logistic Model*

Description

The validate function when used on an object created by lrm 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 corrected slope can be thought of as shrinkage factor that takes into account overfitting.

Usage

```
# fit <- lrm(formula=response ~ terms, x=TRUE, y=TRUE)
## S3 method for class 'lrm':
validate(fit, method="boot", B=40,
         bw=FALSE, rule="aic", type="residual", sls=0.05, aics=0,
         pr=FALSE,  kint, Dxy.method=if(k==1) 'somers2' else 'lrm',
         emax.lim=c(0,1), ...)
```

Arguments

fit	a fit derived by lrm. The options x=TRUE and y=TRUE must have been specified.
method	
B	
bw	
rule	
type	
sls	
aics	
pr	see validate and predab.resample
kint	In the case of an ordinal model, specify which intercept to validate. Default is the middle intercept.
Dxy.method	"lrm" to use lrms computation of D_{xy} correlation, which rounds predicted probabilities to nearest .002. Use Dxy.method="somers2" (the default) to instead use the more accurate but slower somers2 function. This will matter most when the model is extremely predictive. The default is "lrm" for ordinal models, since somers2 only handles binary response variables.

emax.lim	range of predicted probabilities over which to compute the maximum error. Default is entire range.
...	other arguments to pass to lrm.fit (now only maxit and tol are allowed) 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.

Value

a matrix with rows corresponding to D_{xy}, R^2, Intercept, Slope, E_{max}, D, U, Q, amd B, 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 ordinal models, U, Q, B to not appear.

Side Effects

prints a summary, and optionally statistics for each re-fit

Author(s)

Frank Harrell
Department of Biostatistics, Vanderbilt University
f.harrell@vanderbilt.edu

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, Design, Design.trans, calibrate, somers2, cr.setup

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=150
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), 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 the list of resampling methods.

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,
         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	
pr	see validate and predab.resample
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, 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
f.harrell@vanderbilt.edu

See Also

ols, predab.resample, fastbw, Design, Design.trans, calibrate

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=150

#Validate stepwise model with typical (not so good) stopping rule
validate(f, B=20, bw=TRUE, rule="p", sls=.1, type="individual")
```

---

validate.tree *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. This function is a modification of `cv.tree` which should be consulted for details. There are `print` and `plot` methods for objects created by `validate.tree`.

Usage

```
# f <- tree(formula=y ~ x1 + x2 + ...) # or rpart
## S3 method for class 'tree':
validate(fit, method, B, bw, rule, type, sls, aics, pr=TRUE,
    k, rand, xval=10, FUN, ...)
## S3 method for class 'rpart':
validate(fit, ...)
## S3 method for class 'validate.tree':
print(x, ...)
## S3 method for class 'validate.tree':
plot(x, what=c("mse","dxy"), legendloc=locator, ...)
```

Arguments

`fit`	an object created by `tree` or `rpart` or having the same attributes as one created by `tree`. If it was created by `rpart` you must have specified the `model=TRUE` argument to `rpart`.
`method,B,bw,rule,type,sls,aics`	are there only for consistency with the generic `validate` function; these are ignored

x	the result of validate.tree
k	a sequence of cost/complexity values. By default these are obtained from calling FUN with no optional arguments (if tree) or from the rpart cptable object in the original fit object. You may also specify a scalar or vector.
rand	see cv.tree
xval	number of splits
FUN	the name of a function which produces a sequence of trees, such as prune.tree or shrink.tree or prune.rpart. Default is prune.tree for fits from tree and prune.rpart for fits from rpart.
...	additional arguments to FUN (ignored by print,plot). For validate.rpart, ... can be the same arguments used in validate.tree.
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.tree" 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
f.harrell@vanderbilt.edu

See Also

rpart, somers2, rcorr.cens, cv.tree, 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)
library(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)
```

---

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`, or `glm`

Value

vector of vifs

Author(s)

Frank Harrell
Department of Biostatistics
Vanderbilt University
f.harrell@vanderbilt.edu

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

Design.Misc (for num.intercepts and Varcov)

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
f.harrell@vanderbilt.edu

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`, `Design`, `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
```

Index