Package 'LogicReg'

January 20, 2025

Contents

---

cumhaz   *Cumulative hazard transformation*

Description

Transforms survival times using the cumulative hazard function.

Usage

```
cumhaz(y, d)
```

Arguments

y	vector of nonnegative survival times
d	vector of censoring indicators, should be the same length as y. If d is missing the data is assumed to be uncensored.

Value

A vector of transformed survival times.

Note

The primary use of doing a cumulative hazard transformation is that after such a transformation, exponential survival models yield results that are often very much comparable to proportional hazards models. In our implementation of Logic Regression, however, exponential survival models run much faster than proportional hazards models when there are no continuous separate covariates.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

See Also

[logreg](#)

Examples

```
data(logreg.testdat)
#
# this is not survival data, but it shows the functionality
yy <- cumhaz(exp(logreg.testdat[,1]), logreg.testdat[, 2])
# then we would use
# logreg(resp=yy, cens=logreg.testdat[,2], type=5, ...
# insted of
# logreg(resp=logreg.testdat[,1], cens=logreg.testdat[,2], type=4, ...
```

---

eval.logreg   *Evaluate a Logic Regression tree*

Description

This function evaluates a logic tree, typically a part of an object generated by logreg.

Usage

```
eval.logreg(ltree, data)
```

Arguments

ltree   an object of class logregmodel or an object of class logregtree. Typically this object will be part of the result of an object of class logreg, generated with select = 1 (single model fit), select = 2 (multiple model fit), or select = 6 (greedy stepwise fit).

data   a data frame on which the logic tree is to be evaluated. data should be binary, and have the same number of columns as the bin component of the original logreg fit.

Value

A binary vector with length equal to the number of rows of data; a 1 corresponds to cases for which ltree was TRUE and a 0 corresponds to cases for which ltree was FALSE if ltree was an object of class logregtree or the trees component of such an object. Otherwise a matrix with one column for each tree in ltree.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

See Also

logreg, logregtree, logregmodel, frame.logreg, logreg.testdat

Examples

```
data(logreg.savefit1)
# myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 1000)
# logreg.savefit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21],
#                 type = 2, select = 1, ntrees = 2, anneal.control = myanneal)
tree1 <- eval.logreg(logreg.savefit1$model$trees[[1]], logreg.savefit1$binary)
tree2 <- eval.logreg(logreg.savefit1$model$trees[[2]], logreg.savefit1$binary)
alltrees <- eval.logreg(logreg.savefit1$model, logreg.savefit1$binary)
```

| frame.logreg | *Constructs a data frame for one or more Logic Regression models* |

Description

Evaluates all components of one or more Logic Regression models fitted by a single call to logreg.

Usage

```
frame.logreg(fit, msz, ntr, newbin, newresp, newsep, newcens, newweight)
```

Arguments

fit	object of class logreg, that resulted from applying the function logreg with select = 1 (single model fit), select = 2 (multiple model fit), or select = 6 (greedy stepwise fit).
msz	if frame.logreg is executed on an object of class logreg, that resulted from applying the function logreg with select = 2 (multiple model fit) or select = 6 (greedy stepwise fit) all logic trees for all fitted models are returned. To restrict the model size and the number of trees to some models, specify msz and ntr (for select = 2) or just msz (for select = 6).
ntr	see msz.
newbin	binary predictors to evaluate the logic trees at. If newbin is omitted, the original (training) data is used.
newresp	the response. If newbin is omitted, the original (training) response is used. If newbin is specified and newresp is omitted, the resulting data frame will not have a response column.
newsep	separate (linear) predictors. If newbin is omitted, the original (training) predictors are used, even if newsep is specified.
newweight	case weights. If newbin is omitted, the original (training) weights are used. If newbin is specified and newweight is omitted, the weights are taken to be 1.
newcens	censoring indicator. For proportional hazards models and exponential survival models only. If newbin is omitted, the original (training) censoring indicators are used. If newbin is specified and newcens is omitted, the censoring indicators are taken to be 1.

Details

This function calls `eval.logreg`.

Value

A data frame. The first column is the response, later columns are weights, censoring indicator, separate predictors (all of which are only provided if they are relevant) and all logic trees. Column names should be transparent.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

See Also

logreg, eval.logreg, predict.logreg, logreg.testdat

Examples

```
data(logreg.savefit1,logreg.savefit2,logreg.savefit6)
#
# fit a single mode
# myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 1000)
# logreg.savefit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21],
#               type = 2, select = 1, ntrees = 2, anneal.control = myanneal)
frame1 <- frame.logreg(logreg.savefit1)
#
# a complete sequence
# myanneal2 <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 0)
# logreg.savefit2 <- logreg(select = 2, ntrees = c(1,2), nleaves =c(1,7),
#               oldfit = logreg.savefit1, anneal.control = myanneal2)
frame2 <- frame.logreg(logreg.savefit2)
#
# a greedy sequence
# logreg.savefit6 <- logreg(select = 6, ntrees = 2, nleaves =c(1,12), oldfit = logreg.savefit1)
frame6  <- frame.logreg(logreg.savefit6, msz = 3:5) # restrict the size
```

| logreg | *Logic Regression* |

Description

Fit one or a series of Logic Regression models, carry out cross-validation or permutation tests for such models, or fit Monte Carlo Logic Regression models.

Logic regression is a (generalized) regression methodology that is primarily applied when most of the covariates in the data to be analyzed are binary. The goal of logic regression is to find predictors that are Boolean (logical) combinations of the original predictors. Currently the Logic Regression methodology has scoring functions for linear regression (residual sum of squares), logistic regression (deviance), classification (misclassification), proportional hazards models (partial likelihood), and exponential survival models (log-likelihood). A feature of the Logic Regression methodology is that it is easily possible to extend the method to write ones own scoring function if you have a different scoring function. `logreg.myown` contains information on how to do so.

Usage

```
logreg(resp, bin, sep, wgt, cens, type, select, ntrees, nleaves,
        penalty, seed, kfold, nrep, oldfit, anneal.control, tree.control,
        mc.control)
```

Arguments

resp	vector with the response variables. Let n1 be the length of this column.
bin	matrix or data frame with binary data. Let n2 be the number of columns of this object. `bin` should have n1 rows.
sep	(optional) matrix or data frame that is fitted additively in the logic regression model. `sep` should have n1 rows. When exponential survival models (`type = 5`) are used, the additive predictors have to be binary. When logistic regression models (`type = 3`) are used `logreg` is much faster when all additive predictors are binary.
wgt	(optional) vector of length n1 with case weights; default is `rep(1,n1)`.
cens	(optional) an indicator variable with censoring indicators if `type` equals 4 (proportional hazards model) or 5 (exponential survival model); default is `rep(1,n1)`.
type	type of model to be fit: (1) classification, (2) regression, (3) logistic regression, (4) proportional hazards model (Cox regression), (5) exponential survival model, or (0) your own scoring function. If `type = 0`, the code needs to be recompiled, uncompiled `type = 0` results in a constant score of 0, which may be useful to generate a sample from the prior when `select = 7` (Monte Carlo Logic Regression).
select	type of model selection to be carried out: (1) fit a single model, (2) fit multiple models, (3) cross-validation, (4) null-model permutation test, (5) conditional permutation test, (6) a greedy stepwise algorithm, or (7) Monte Carlo Logic Regression (using MCMC). See details below.

ntrees	number of logic trees to be fit. A single number if you select to fit a single model (select = 1), carry out the null-model permutation test (select = 4), carry out greedy stepwise selection (select = 6) or, select using MCMC (select = 7), or a range (e.g. c(ntreeslow,ntreeshigh)) for any of the other selection options. In our applications, we usually ended up with models having between one and four trees. In general, fitting one and two trees in the initial exploratory analysis is a good idea.
nleaves	maximum number of leaves to be fit in all trees combined. A single number if you select to fit a single model (select = 1) carry out the null-model permutation test (select = 4), carry out greedy stepwise selection (select = 6) or, select using MCMC (select = 7), or a range (e.g. c(nleaveslow,nleaveshigh)) for any of the other selection options. If select is 1, 4, 6, or 7, the default is -1, which is expanded to become ntrees * tree.control\$treesize .
penalty	specifying the penalty parameter allows you to penalize the score of larger models. The penalty takes the form penalty times the number of leaves in the model. (For some score functions, we compute average scores: the penalty is naturally adjusted.) Thus penalty = 2 is somewhat comparable to AIC. Note, however, that there is no relation between the size of the model and the number of parameters (dimension) of the model, as is usual the case for AIC like penalties. penalty is only relevant when select = 1.
seed	a seed for the random number generator. The random seed is taken to be abs(seed). \ For the cross-validation version, if seed < 0 the sequence of the cases is not permuted for division in training-test sets. This is useful if you already permuted the sequence, and wish to compare results with other approaches, or if there is a relation between the sequence of the cases, for example for a matched case-control study.
kfold	the number of groups the cases are randomly assigned to. In turn, the model is trained on (kfold - 1) of those groups, and scored on the group left out. Common choices are kfold = 5 and kfold = 10. Only relevant for cross-validation (select = 3).
nrep	the number of runs on permuted data for each model size. We recommend first running this program with a small number of repetitions (e.g. 10 or 25) before sending off a big job. Only relevant for the null-model test (select = 4) or the permutation test (select = 5).
oldfit	object of class logreg, typically the result of a previous call to logreg. All options that are not specified default to the value used in oldfit. For the permutation test (select = 5) an oldfit object obtained with select = 2 (fit multiple models) is mandatory, as the best models of each size need to be in place.
anneal.control	simulated annealing parameters - best set using the function logreg.anneal.control.
tree.control	several secondary parameters - best set using the function logreg.tree.control.
mc.control	Markov chain Monte Carlo parameters - best set using the function logreg.mc.control.

Details

Logic Regression is an adaptive regression methodology that attempts to construct predictors as Boolean combinations of binary covariates.

In most regression problems a model is developed that only relates the main effects (the predictors or transformations thereof) to the response. Although interactions between predictors are considered sometimes as well, those interactions are usually kept simple (two- to three-way interactions at most). But often, especially when all predictors are binary, the interaction between many predictors is what causes the differences in response. This issue often arises in the analysis of SNP microarray data or in data mining problems. Given a set of binary predictors X, we try to create new, better predictors for the response by considering combinations of those binary predictors. For example, if the response is binary as well (which is not required in general), we attempt to find decision rules such as "if X1, X2, X3 and X4 are true", or "X5 or X6 but not X7 are true", then the response is more likely to be in class 0. In other words, we try to find Boolean statements involving the binary predictors that enhance the prediction for the response. In more specific terms: Let X1,...,Xk be binary predictors, and let Y be a response variable. We try to fit regression models of the form g(E[Y]) = b0 + b1 L1+ ...+ bn Ln, where Lj is a Boolean expression of the predictors X, such as Lj=[(X2 or X4c) and X7]. The above framework includes many forms of regression, such as linear regression (g(E[Y])=E[Y]) and logistic regression (g(E[Y])=log(E[Y]/(1-E[Y]))). For every model type, we define a score function that reflects the "quality" of the model under consideration. For example, for linear regression the score could be the residual sum of squares and for logistic regression the score could be the deviance. We try to find the Boolean expressions in the regression model that minimize the scoring function associated with this model type, estimating the parameters bj simultaneously with the Boolean expressions Lj. In general, any type of model can be considered, as long as a scoring function can be defined. For example, we also implemented the Cox proportional hazards model, using the partial likelihood as the score.

Since the number of possible Logic Models we can construct for a given set of predictors is huge, we have to rely on some search algorithms to help us find the best scoring models. We define the move set by a set of standard operations such as splitting and pruning the tree (similar to the terminology introduced by Breiman et al (1984) for CART). We investigated two types of algorithms: a greedy and a simulated annealing algorithm. While the greedy algorithm is very fast, it does not always find a good scoring model. The simulated annealing algorithm usually does, but computationally it is more expensive. Since we have to be certain to find good scoring models, we usually carry out simulated annealing for our case studies. However, as usual, the best scoring model generally over-fits the data, and methods to separate signal and noise are needed. To assess the over-fitting of large models, we offer the option to fit a model of a specific size. For the model selection itself we developed and implemented permutation tests and tests using cross-validation. If sufficient data is available, an analysis using a training and a test set can also be carried out. These tests are rather complicated, so we will not go into detail here and refer you to Ruczinski I, Kooperberg C, LeBlanc ML (2003), cited below.

There are two alternatives to the simulated annealing algorithm. One is a stepwise greedy selection of models. This is when setting `select = 6`, and yields a sequence of models from size 1 through a maximum size. At each time among all the models that are one larger than the current model the best model is selected, yielding a sequence of models of different sizes. Usually these models are not the best possible, and, if the simulated annealing chain is long enough, you should expect that the models selected using `select = 2` are better.

The second alternative is to run a Markov Chain Monte Carlo (MCMC) algorithm. This is what is done in Monte Carlo Logic Regression. The algorithm used is a reversible jump MCMC algorithm, due to Green (1995). Other than the length of the Markov chain, the only parameter that needs to be set is a parameter for the geometric prior on model size. Other than in many MCMC problems, the goal in Monte Carlo Logic Regression is not to yield one single best predicting model, but rather to provide summaries of all models. These are exactly the elements that are shown above as the output

when `select = 7`.

MONITORING

The help file for `logreg.anneal.control`, contains more information on how to monitor the simulated annealing optimization for logreg. Here is some general information.

Find the best scoring model of any size (`select = 1`)

During the iterations the following information is printed out:

log-temp	current score	best score	acc /	rej /	sing	current parameters
-1.000	1.494	1.494	0(0)	0	0	2.88 -1.99 0.00
-1.120	1.150	1.043	655(54)	220	71	3.63 0.15 -1.82
-1.240	1.226	1.043	555(49)	316	80	3.83 0.05 -1.71
...						
-2.320	0.988	0.980	147(36)	759	58	3.00 -2.11 1.11
-2.440	0.982	0.980	25(31)	884	60	2.89 -2.12 1.24
-2.560	0.988	0.979	35(61)	850	51	3.00 -2.11 1.11
...						
-3.760	0.964	0.964	2(22)	961	15	2.57 -2.15 1.55
-3.880	0.964	0.964	0(17)	961	22	2.57 -2.15 1.55
-4.000	0.964	0.964	0(13)	970	17	2.57 -2.15 1.55

log-temp:
logarithm (base 10) of the temperature at the last iteration before the print out.
current score:
the score after the last iterations.
best score:
the single lowest score seen at any iteration.
acc:
the number of proposed moves that were accepted since the last print out for which the model changed, within parenthesis, the number of those that were identical in score to the move before acceptance.
rej:
the number of proposed moves that gave numerically acceptable results, but were rejected by the simulated annealing algorithm since the last print out.
sing:
the number of proposed moves that were rejected because they gave numerically unacceptable results, for example because they yielded a singular system.
current parameters:
the values of the coefficients (first for the intercept, then for the linear (separate) components, then for the logic trees).

This information can be used to judge the convergence of the simulated annealing algorithm, as described in the help file of `logreg.anneal.control`. Typically we want (i) the number of acceptances to be high in the beginning, (ii) the number of acceptances with different scores to be low at the end, and (iii) the number of iterations when the fraction of acceptances is moderate to be as large as possible.

Find the best scoring models for various sizes (`select = 2`)

During the iterations the same information as for *find the best scoring model of any size*, for each size model considered.

Carry out cross-validation for model selection (`select = 3`)

Information about the simulated annealing as described above can be printed out. Otherwise, during the cross-validation process information like

 `Step 5 of 10 [ 2 trees; 4 leaves] The CV score is 1.127 1.120 1.052 1.122`

The first of the four scores is the *training*-set score on the current validation sample, the second score is the average of all the *training*-set scores that have been processed for this model size, the third is the *test*-set score on the current validation sample, and the fourth score is the average of all the *test*-set scores that have been processed for this model size. Typically we would prefer the model with the lowest test-set score average over all cross-validation steps.

Carry out a permutation test to check for signal in the data (`select = 4`)

Information about the simulated annealing as described above can be printed out. Otherwise, first the score of the model of size 0 (typically only fitting an intercept) and the score of the best model are printed out. Then during the permutation lines like

 `Permutation number 21 out of 50 has score 1.47777`

are printed. Each score is the result of fitting a logic tree model, on data where the response has been permuted. Typically we would believe that there is signal in the data if most permutations have worse (higher) scores than the best model, but we may believe that there is substantial over-fitting going on if these permutation scores are much better (lower) than the score of the model of size 0.

Carry out a permutation test for model selection (`select = 5`)

To be able to run this option, an object of class `logreg` that was run with (`select = 2`) needs to be in place. Information about the simulated annealing as described above can be printed out. Otherwise, lines like

 `Permutation number 8 out of 25 has score 1.00767 model size 3 with 2 tree(s)`

are printed. We can compare these scores to the tree of the same size and the best tree. If the scores are about the same as the one for the best tree, we think that the "true" model size may be the one that is being tested, while if the scores are much worse, the true model size may be larger. The comparison with the model of the same size suggests us again how much over-fitting may be going on. `plot.logreg` generates informative histograms.

Greedy stepwise selection of Logic Regression models (`select = 6`)

The scores of the best greedy models of each size are printed.

Monte Carlo Logic Regression (`select = 7`)

A status line is printed every so many iterations. This information is probably not very useful, other than that it helps you figure out how far the code is.

PARAMETERS

As Logic Regression is written in Fortran 77 some parameters had to be hard coded in. The default values of these parameters are

maximum number of columns in the input file: 1000
maximum number of leaves in a logic tree: 128
maximum number of logic trees: 5
maximum number of separate parameters: 50

maximum number of total parameters(separate + trees): 55

If these parameters are not large enough (an error message will let you know this), you need to reset them in **slogic.f** and recompile. In particular, the statements defining these parameters are

```
PARAMETER (LGCn2MAX = 1000)
PARAMETER (LGCnknMAX = 128)
PARAMETER (LGCntrMAX = 5)
PARAMETER (LGCnsepMAX = 50)
PARAMETER (LGCbetaMAX = 55)
```

The unfortunate thing is that you will have to change these parameter definitions several times in the code. So search until you have found them all.

Value

An object of the class logreg. This contains virtually all input parameters, and in addition

If select = 1:

an object of class logregmodel: the Logic Regression model. This model contains a list of ntrees objects of class logregtree.

If select = 2 or select = 6:

nmodels: the number of models fitted.
allscores: a matrix with 3 columns, containing the scores of all models. Column 1 contains the score, column 2 the number of leaves and column 3 the number of trees.
alltrees: a list with nmodels objects of class logregmodel.

If select = 3:

cvscores: a matrix with the results of cross validation. The train.ave and test.ave columns for train and test contain running averages of the scores for individual validation sets. As such these scores are of most interest for the rows where k=kfold.

If select = 4:

nullscore: score of the null-model.
bestscore: score of the best model.
randscores: scores of the permutations; vector of length nrep.

If select = 5:

bestscore: score of the best model.
randscores: scores of the permutations; each column corresponds to one model size.

If select = 7:

size: a matrix with two columns, indicating which size models were fit how often.
single: a vector with as many elements as there are binary predictors. single[i] shows how often

predictor i is in any of the MCMC models. Note that when a predictor is twice in the same model, it is only counted once, thus, in particular, sum(size[,1]*size[,2] will typically be slightly larger than sum(single).

double: square matrix with as size the number of binary predictors. double[i,j] shows how often predictors i and j are in the same tree of the same MCMC model if i>j, if i<=j double[i,j] equals zero. Note that for models with several logic trees two predictors can both be in the model but not be in the same tree.

triple: square 3D array with as size the number of binary predictors. See double, but here triple[i,j,k] shows how often three predictors are jointly in one logic tree.

In addition, the file slogiclisting.tmp in the current working directory can be created. This file contains a compact listing of all models visited. Column 1: proportional to the log posterior probability of the model; column 2: score (log-likelihood); column 3: how often was this model visited, column 4 through 3 + maximum number of leaves: summary of the first tree, if there are two trees, column 4 + maximum number of leaves through 3 + twice the maximum number of leaves contains the second tree, and so on.

In this compact notation, leaves are in the same sequence as the rows in a logregtree object; a zero means that the leave is empty, a 1000 means an "and" and a 2000 an "or", any other positive number means a predictor and a negative number means "not" that predictor.

The mc.control element output can be used to surppress the creation of double, triple, and/or slogiclisting.tmp. There doesn't seem to be too much use in surppressing double. Surpressing triple speeds up computations a bit (in particular on machines with limited memory when there are many binary predictors), and reduces the size of both the code and the object, surppressing slogiclisting.tmp saves the creation of a possibly very large file, which can slow down the code considerably. See logreg.mc.control for details.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Kooperberg C, Ruczinski I, LeBlanc ML, Hsu L (2001). Sequence Analysis using Logic Regression, *Genetic Epidemiology*, **21**, S626-S631.

Kooperberg C, Ruczinki I (2005). Identifying interacting SNPs using Monte Carlo Logic Regression, *Genetic Epidemiology*, **28**, 157-170.

Selected chapters from the dissertation of Ingo Ruczinski, available from https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf

See Also

eval.logreg, frame.logreg, plot.logreg, print.logreg, predict.logreg, logregtree, plot.logregtree, print.logregtree, logregmodel, plot.logregtree, print.logregtree, logreg.myown, logreg.anneal.control, logreg.tree.control, logreg.mc.control, logreg.testdat

Examples

```
data(logreg.savefit1,logreg.savefit2,logreg.savefit3,logreg.savefit4,
     logreg.savefit5,logreg.savefit6,logreg.savefit7,logreg.testdat)

myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 500, update = 100)
# in practie we would use 25000 iterations or far more - the use of 500 is only
# to have the examples run fast
## Not run: myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 500)
fit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21], type = 2,
                select = 1, ntrees = 2, anneal.control = myanneal)
# the best score should be in the 0.95-1.10 range
plot(fit1)
# you'll probably see X1-X4 as well as a few noise predictors
# use logreg.savefit1 for the results with 25000 iterations
 plot(logreg.savefit1)
 print(logreg.savefit1)
 z <- predict(logreg.savefit1)
 plot(z, logreg.testdat[,1]-z, xlab="fitted values", ylab="residuals")
# there are some streaks, thanks to the very discrete predictions
#
# a bit less output
 myanneal2 <- logreg.anneal.control(start = -1, end = -4, iter = 500, update = 0)
# in practie we would use 25000 iterations or more - the use of 500 is only
# to have the examples run fast
## Not run: myanneal2 <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 0)
#
# fit multiple models
 fit2 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21], type = 2,
                select = 2, ntrees = c(1,2), nleaves =c(1,7), anneal.control = myanneal2)
# equivalent
## Not run: fit2 <- logreg(select = 2, ntrees = c(1,2), nleaves =c(1,7), oldfit = fit1,
                 anneal.control = myanneal2)
## End(Not run)
 plot(fit2)
# use logreg.savefit2 for the results with 25000 iterations
 plot(logreg.savefit2)
 print(logreg.savefit2)
# After an initial steep decline, the scores only get slightly better
# for models with more than four leaves and two trees.
#
# cross validation
 fit3 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21], type = 2,
                 select = 3, ntrees = c(1,2), nleaves=c(1,7), anneal.control = myanneal2)
# equivalent
## Not run: fit3 <- logreg(select = 3, oldfit = fit2)
 plot(fit3)
# use logreg.savefit3 for the results with 25000 iterations
 plot(logreg.savefit3)
# 4 leaves, 2 trees should top
# null model test
 fit4 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21], type = 2,
                select = 4, ntrees = 2, anneal.control = myanneal2)
```

```
# equivalent
## Not run: fit4 <- logreg(select = 4, anneal.control = myanneal2, oldfit = fit1)
 plot(fit4)
# use logreg.savefit4 for the results with 25000 iterations
plot(logreg.savefit4)
# A histogram of the 25 scores obtained from the permutation test. Also shown
# are the scores for the best scoring model with one logic tree, and the null
# model (no tree). Since the permutation scores are not even close to the score
# of the best model with one tree (fit on the original data), there is overwhelming
# evidence against the null hypothesis that there was no signal in the data.
 fit5 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21], type = 2,
           select = 5, ntrees = c(1,2), nleaves=c(1,7), anneal.control = myanneal2,
           nrep = 10, oldfit = fit2)
# equivalent
## Not run: fit5 <- logreg(select = 5, nrep = 10, oldfit = fit2)
 plot(fit5)
# use logreg.savefit5 for the results with 25000 iterations and 25 permutations
 plot(logreg.savefit5)
# The permutation scores improve until we condition on a model with two trees and
# four leaves, and then do not change very much anymore. This indicates that the
# best model has indeed four leaves.
#
# greedy selection
 fit6 <- logreg(select = 6, ntrees = 2, nleaves =c(1,12), oldfit = fit1)
 plot(fit6)
# use logreg.savefit6 for the results with 25000 iterations
 plot(logreg.savefit6)
#
# Monte Carlo Logic Regression
fit7 <- logreg(select = 7, oldfit = fit1, mc.control=
               logreg.mc.control(nburn=500, niter=2500, hyperpars=log(2), output=-2))
# we need many more iterations for reasonable results
## Not run: logreg.savefit7 <- logreg(select = 7, oldfit = fit1, mc.control=
               logreg.mc.control(nburn=1000, niter=100000, hyperpars=log(2)))
## End(Not run)
#
plot(fit7)
# use logreg.savefit7 for the results with 100,000 iterations
plot(logreg.savefit7)
```

---

logreg.anneal.control   *Control for Logic Regression*

Description

Control of simulated annealing parameters needed in logreg.

Usage

```
logreg.anneal.control(start=0, end=0, iter=0, earlyout=0, update=0)
```

Arguments

start
: the upper temperature (on a log10 scale) in the annealing chain. I.e. if `start` = 3, the annealing chain starts at temperature 1000. The acceptance function is the usual `min(1,exp(-diff(scores)/temp))`, so any temperature larger than what would be expected as possible differences between any two models pretty much generates a random walk in the beginning, and means that you need to wait longer on results. A too low starting temperature means that the chain may end up in a local optimal (rather than global optimal) solution. If you select both `start` and `end` the default of 0, the program will attempt to find reasonable numbers itself (it is known to be only moderately successful in this though).

end
: the lower temperature (on a log10 scale) in the annealing chain. I.e. if `end` is -2, the annealing chain ends at temperature 0.01. If this temperature is very low one can use the early out possibility listed below, as otherwise the chain may run longer than desired!

iter
: the total number of iterations in the annealing chain. This is the total over all annealing chains, not the number of iterations of a chain at a given temperature. If this number is too small the chain may not find a good (the best) solution, if the chain is too long the program may take long...

earlyout
: if the end temperature is very low, the simulated annealing algorithm may not move any more, but one still needs to wait on all possible moves being evaluated (and rejected)! An early out possibility is offered. If during consecutive five blocks of `earlyout` iterations, in each block 10 or fewer moves are accepted (for which the score changes), the program terminates. This is a desirable option after one is convinced the program otherwise runs fine: it can be dangerous on the first run.

update
: every how many iterations there should be an update of the scores. I.e. if `update` = 1000, a score will get printed every 1000 iterations. So if `iter` = 100000 iterations, there will be 100 updates on your screen. If you `update` = 0, a one line summary for each fitted model is printed. If `update` = -1, there is virtually no printed output.

Details

Missing arguments take defaults. If the argument `start` is a list with arguments `start`, `end`, `iter`, `earlyout`, and `update`, those values take precedent of directly specified values.

This is a rough outline how the automated simulated annealing works: The algorithm starts running at a very high temperature, and decreases the temperature until the acceptance ratio of moves is below a certain threshold (in the neighborhood of 95%). At this point we run longer chains at fixed temperatures, and stop the search when the last "n" consecutive moves have been rejected. If you think that the search was either not sufficiently long or excessively long (both of which can very well happen since it is pretty much impossible to specify default values that are appropriate for all sorts of data and models), you can over-write the default values.

If you want more detailed information continue reading....

These are some more detailed suggestions on how to set the parameters for the beginning temperature, end temperature and number of iterations for the Logic Regression routine. Note that if `start` temperature and end temperature are both zero, the routine uses its default values. The number of

iterations `iter` is irrelevant in this case. In our opinion, the default values are OK, but not great, and you can usually do better if you're willing to invest time in learning how to set the parameters.

The starting temperature is the log(10) value of `start` - i.e., if `start` is 2 it means iterations start at a temperature of 100. The end temperature is again the log(10) value. The number of iterations are equidistant on a log-scale.

Considerations in setting these parameters.....

1) `start` temperature. If this is too high you're "wasting time", as the algorithm is effectively just making a random walk at high temperatures. If the starting temperature is too low, you may already be in a (too) localized region of the search space, and never reach a good solution. Typically a starting temperature that gives you 90% or so acceptances (ignoring the rejected attempts, see below) is good. Better a bit too high than too low. But don't waste too much time.

2) end temperature. By the time that you reach the end temperature the number of accepted iterations should be only a few per 1000, and the best score should no longer change. Even zero acceptances is fine. If there are many more acceptances, lower end. If there are zero acceptances for many cycles in a row, raise it a bit. You can set a lower end temperature than needed using the `earlyout` test: if in 5 consecutive cycles of 1000 iterations there are fewer than a specified number of acceptances per cycle, the program terminates.

3) number of iterations. What really counts is the number of iterations in the "crunch time", when the number of acceptances is, say, more than 5% but fewer than 40% of the iterations. If you print summary statistics in blocks of 1000, you want to see as many blocks with such acceptance numbers as possible. Obviously within what is reasonable.

Here are two examples, with my analysis....

(A) `logreg.anneal.control(start = 2, end = 1, iter = 50000, update = 1000)`

The first few lines are (cutting of some of the last columns...)

log-temp	current score	best score	acc /	rej /	sing	current parameters
2.000	1198.785	1198.785	0	0	0	0.508 -0.368 -0.144
1.980	1197.962	1175.311	719(18)	34	229	1.273 -0.275 -0.109
1.960	1197.909	1168.159	722(11)	38	229	0.416 -0.345 -0.173
1.940	1181.545	1168.159	715(19)	35	231	0.416 -0.345 -0.173
...						
1.020	1198.258	1167.578	663(16)	128	193	1.685 -0.216 -0.024
1.000	1198.756	1167.578	641(23)	104	232	1.685 -0.216 -0.024
1.000	1198.756	1167.578	1(0)	0	0	1.685 -0.216 -0.024

Ignore the last line. This one is just showing a refitting of the best model. Otherwise, this suggests (i) end is ***way*** too high, as there are still have more than 600 acceptances in blocks of 1000. It is hard to judge what end should be from this run. (ii) The initial number of acceptances is really high (719+18)/(719+18+34))=95% - but when 1.00 is reached it's at about 85%. One could change `start` to 1, or keep it at 2 and play it save.

(B) `logreg.anneal.control(start = 2, end = -2, iter = 50000, update = 1000)` - different dataset/problem

The first few lines are

log-temp	current score	best score	acc /	rej /	sing	current parameters

2.000	1198.785	1198.785	0(0)	0	0	0.50847	-0.36814
1.918	1189.951	1172.615	634(23)	22	322	0.38163	-0.28031
1.837	1191.542	1166.739	651(24)	32	293	1.75646	-0.22451
1.755	1191.907	1162.902	613(30)	20	337	1.80210	-0.32276

The last few are

log-temp	current score	best score	acc /	rej /	sing	current parameters	
-1.837	1132.731	1131.866	0(18)	701	281	0.00513	-0.45994
-1.918	1132.731	1131.866	0(25)	676	299	0.00513	-0.45994
-2.000	1132.731	1131.866	0(17)	718	265	0.00513	-0.45994
-2.000	1132.731	1131.866	0(0)	0	1	0.00513	-0.45994

But there really weren't any acceptances since

log-temp	current score	best score	acc /	rej /	sing	current parameters	
-0.449	1133.622	1131.866	4(21)	875	100	0.00513	-0.45994
-0.531	1133.622	1131.866	0(19)	829	152	0.00513	-0.45994
-0.612	1133.622	1131.866	0(33)	808	159	0.00513	-0.45994

Going down from 400 to fewer than 10 acceptances went pretty fast....

log-temp	current score	best score	acc /	rej /	sing	current parameters	
0.776	1182.156	1156.354	464(31)	258	247	1.00543	-0.26602
0.694	1168.504	1150.931	306(17)	355	322	1.56695	-0.43351
0.612	1167.747	1150.931	230(38)	383	349	1.56695	-0.43351
0.531	1162.085	1145.920	124(12)	571	293	1.15376	-0.15223
0.449	1143.841	1142.321	63(20)	590	327	2.20150	-0.43795
0.367	1176.152	1142.321	106(21)	649	224	2.20150	-0.43795
0.286	1138.384	1131.866	62(18)	731	189	0.00513	-0.45994
0.204	1138.224	1131.866	11(27)	823	139	0.00513	-0.45994
0.122	1150.370	1131.866	15(12)	722	251	0.00513	-0.45994
0.041	1144.536	1131.866	30(19)	789	162	0.00513	-0.45994
-0.041	1137.898	1131.866	21(25)	911	43	0.00513	-0.45994
-0.122	1139.403	1131.866	12(30)	883	75	0.00513	-0.45994

What does this tell me - (i) start was probably a bit high - no real harm done, (ii) end was lower than needed. Since there really weren't any acceptances after 10log(T) was about (-0.5), an ending log-temperature of (-1) would have been fine, (iii) there were far too few runs. The crunch time didn't take more than about 10 cycles (10000 iterations). You see that this is the time the "best model" decreased quite a bit - from 1156 to 1131. I would want to spend considerably more than 10000 iterations during this period for a larger problem (how many depends very much on the size of the problem). So, I'd pick (A)logreg.anneal.control(start = 2,end = -1, iter = 200000,

update = 5000). Since the total range is reduced from 2-(-2)=4 to 2-(-1)=3, over a range of 10log temperatures of 1 there will be 200000/3=67000 rather than 50000/4=12500 iterations. I would repeat this run a couple of times.

In general I may sometimes run several models, and check the scores of the best models. If those are all the same, I'm very happy, if they're similar but not identical, it's OK, though I may run one or two longer chains. If they're very different, something is wrong. For the permutation test and cross-validation I am usually less picky on convergence.

Value

A list with arguments `start`, `end`, `iter`, `earlyout`, and `update`, that can be used as the value of the argument `anneal.control` of `logreg`.

Author(s)

Ingo Ruczinski `<ingo@jhu.edu>` and Charles Kooperberg `<clk@fredhutch.org>`.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Selected chapters from the dissertation of Ingo Ruczinski, available from `https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf`

See Also

`logreg`, `logreg.mc.control`, `logreg.tree.control`

Examples

```
myannealcontrol <- logreg.anneal.control(start = 2, end = -2, iter = 50000, update = 1000)
```

---

logreg.mc.control   *Control for Logic Regression*

Description

Control of MCMC annealing parameters needed in `logreg`.

Usage

```
logreg.mc.control(nburn=1000, niter=25000, hyperpars=0, update=0,
        output=4)
```

Arguments

nburn	number of burn in MCMC iterations that are ignored when computing summaries
niter	number of MCMC iterations that are used to compute summary statistics
hyperpars	hyperparameters. The code allows up to 10 such parameters, but currently only one is used. In particular, log(P(size=k)/P(size=k+1)) equals hyperpars[1], where P is the prior on model size. Since a maximum model size (specified in logreg is being used, hyperpars[1] can even be smaller than 0.
update	every how many iterations there should be an update of the scores. I.e. if update = 1000, a score will get printed every 1000 iterations. So if iter = 100000 iterations, there will be 100 updates on your screen. If update = 0, a one line summary for each fitted model is printed. If update = -1, there is virtually no printed output.
output	If abs(output) > 1 bivariate statistics are gathered, if abs(output) > 2 trivariate statistics are also gathered, otherwise only univariate statistics are gathered. If output > 0 all fitted models are saved in a text file "slogiclisting.tmp", if output < 0 this does not happen.

Details

Considerations for setting nburn and niter are as for any MCMC problem. In our experience Logic Regression mixes quickly, and a real small nburn (1000, for example) suffices. If there are many trees and large models niter may need to be large.

A more detailed description of the output options can be found in the helpfile of logreg.

Value

A list with arguments nburn, niter, hyperpars, update, and output, that can be used as the value of the argument mc.control of logreg.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Kooperberg C, Ruczinki I (2005). Identifying interacting SNPs using Monte Carlo Logic Regression, *Genetic Epidemiology*, **28**, 157-170.

See Also

[logreg](), [logreg.tree.control](), [logreg.anneal.control]()

Examples

```
mymccontrol <- logreg.mc.control(nburn = 500, niter = 500000, update = 25000,
hyperpars = log(2), output = -2)
```

---

logreg.myown   *Writing your own Logic Regression scoring function*

Description

Help file for writing your own scoring function for `logreg`!

Usage

```
logreg.myown()
```

Details

You can write your own scoring function for `logreg`! This may be useful if you have a model other than those which we already programmed in.

Essentially you need to provide two routines in the file **Myownscoring.f**:
(i) A routine *Myownfitting* which fits your model: it provides a coefficient (beta) for each of the logic trees and provides a score of how good the model is. Low scores are good. (So add a minus sign if your score is a log-likelihood.)
(ii) A routine *Myownscoring* which - given the betas - provides the score of your model. [If you don't use cross-validation, this second routine is not needed, though some dummy routine to satisfy the compiler should still be provided.]

After recompilation, you can fit your model using the option `type = 0` in `logreg`. Below we give an example for a version of the My.own functions for conditional logistic regression which are also provided as **inst/condlogic.ff** when you downloaded the files.

PROGRAMMING DETAILS

Below is a list of variables that are passed on. Most of them are as you expect - response, predictors (binary ones and continuous ones), number of cases, number of predictors. In addition there are two columns - `dcph` and `weight` - that can either be used to pass on an auxiliary variable for each case (discrete for `dcph` and continuous for `weight`), or even some overall auxiliary variables - as these numbers are not used anywhere else. If you do not need any of the variables - just ignore them!

`prtr`:
the predictions of the logic trees in the current model: this is an integer matrix of size n1 times `ntr` - although only the first `nop` columns contain useful information.
`rsp`:
the response variable: this is a real (single precision) vector of length n1.
`dcph`:
censor times: this is an integer vector of length n1 this could be used as an auxiliary (integer) vector - as it is just passed on. (There is no check that this is a 0/1 variable, when you use your own scoring function.) For example, you could use this to pass on something like cluster membership.
`weight`:

weights for the cases this is a real vector of length n1. this could be used as an auxiliary (real) vector - as it is just passed on. There is no check that these numbers are positive, when you choose your own scoring function.
ordrs:
the order (by response size) of the cases This is an integer vector of length n1. For the case with the smallest response this one is 1, for the second smallest 2, and so on. Ties are resolved arbitrary. Always computed, although only used for proportional hazards models. Use it as you wish.
n1:
the total number of cases in the data.
ntr:
the number of logic trees ALLOWED in the tree.
nop:
the number of logic trees in the CURRENT model. The subroutines should work if nop is 0.
wh:
the index of the tree that has been edited in the last move - i.e. the column of prtr that has changes since the last call.
nsep:
number of variables that get fit a separate parameter The subroutines should work if nsep is 0.
seps:
array of the above variables - this is a single precision matrix of size nsep times n1. Note that seps and prtr are stored in different directions.

For *Myownfitting* you should return:
betas:
a vector of parameters of the model that you fit. betas(0) should be the parameter for the intercept betas(1:nsep) should be the parameters for the continuous variables in seps betas((nsep+1):(nsep+nop)) should be the parameters for the binary trees in prtr if you have more parameters, use dcph, or weight; these variables will not be printed however.
score:
whatever score you assign to your model small should be good (i.e. deviance or -log.likelihood).
reject:
an indicator whether or not to reject the proposed move *regardless* of the score (for example when an iteration necessary to determine the score failed to converge (0 = move is OK ; 1 = reject move) set this one to 0 if there is no such condition.

You are allowed to change the values of dcph, and weight.

For *Myownscoring* additional input is:
betas: the coefficients

You should return:
score: whatever score you assign to your model small should be good (i.e. deviance or -log.likelihood). If the model "crashes", you should simply return a very large number.

While we try to prevent that models are singular, it is possible that for your model a single or degenerate model is passed on for evaluation. For Myownfitting you can pass the model back with reject = 1, for Myownscoring you can pass it on with a very large value for score. Currently Myownscoring.f contains empty frames for the scoring functions; condlogic.ff contains an example with conditional logistic regression.

The logic regression program is written in Fortran 77.

CONDITIONAL LOGISTIC REGRESSION

A function for a conditional logistic regression score function is attached as an example function on how to write your own scoring function for Logic Regression. Obviously you can also use it if you have conditional logistic data.

Conditional logistic regression is common model fitting technique for matched case-control studies, in which each case is matched with one or more controls. (In conditional logistic regression several cases could be matched to several controls, in the implementation provided here only one case can be matched with each group of controls.) Conditional logistic regression models are parameterized like regular logistic regression models, except that the intercept is no longer identifiable. See, for example, Breslow and Day - Volume 1 (1990, Statistical Methods in Cancer Research, International Agency for Research on Cancer, Lyon) for details. Conditional logistic regression models are most easily fit using a stratified proportional hazards model (if there is one-to-one case-control matching it can also be fit using logistic regression, but that method breaks down if there is more than one control per case). Each group of a case and controls is one stratum. All cases get an arbitrarily event time of 1.00, and all controls get a censoring time of 2.00.

In our implementation we use the response column to indicate the matching. For all controls this column is 0, for a case it is k, indicating that the next k records are the matched controls for the current case. Thus, we order our cases so that each case is followed by its controls. Cases with a negative response are put in a stratum -1, which is not used in any computations. This has implications for cross-validation. See below.

In *Myownfitting* and *Myownscoring* we first allocate various vectors (strata, index, censoring variable) that are local, as well as some work arrays that are used by our fitting routines. (We need to set some of the parameters for that, see the help page of `logreg` for details.) We then define `idx(j)=j` for `j=1,n1`, and we define the `strata` and `delta` vectors. We use slightly modified versions of the proportional hazards routines that are already used otherwise in the Logic Regression program, to include stratification. After the model is fitted, we assign minus the partial likelihood to `score(1)` and (for *Myownfitting*) we pass on the betas.

Recompile after replacing *Myownscoring.f* by *condlogic.ff*

The permutation and null model versions are not directly usable (we could do some permutation tests, but they require more programming), but we can use cross-validation. Obviously we should keep cases and controls match. To that extend, we would run permutation with a negative seed (see `logreg`) and we would take care ourselves that case-control groups are in a random order, and that every block has the same number of records. We achieve the later by adding some records with response -1. In particular, suppose that we have 19 pairs of case- (single) control data, and that we want to do 3-fold cross validation. We would permute the sequence of the 19 pairs, and add two records with response -1 after the 13th pair, and two records with -1 at the end of the file, so that the total data file would have 42 records.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M.

Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Kooperberg C, Ruczinski I, LeBlanc ML, Hsu L (2001). Sequence Analysis using Logic Regression, *Genetic Epidemiology*, **21**, S626-S631.

Selected chapters from the dissertation of Ingo Ruczinski, available from `https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf`

See Also

logreg

Examples

```
logreg.myown()     # displays this help file
help(logreg.myown) # equivalent
```

---

logreg.savefit1	*Sample results for Logic Regression*

Description

The logreg.savefit objects are the results of fitting logreg with various options. The examples in the functions of the LogicReg packages all use far fewer iterations than is needed. (The number of iterations was reduced to provide quick results for bug-checking.) The number of iterations in the logreg.savefit objects are more reasonable (though they would still be small for larger problems). Otherwise the arguments used to fit the logreg.savefit objects are identical as those used in the examples of logreg. The logreg.savefit objects are used for examples involving things like plotting, printing, and predicting.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>

See Also

logreg, logreg.testdat

Examples

```
data(logreg.savefit1)
print(logreg.savefit1$call)
data(logreg.savefit2)
print(logreg.savefit2$call)
data(logreg.savefit3)
print(logreg.savefit3$call)
data(logreg.savefit4)
print(logreg.savefit4$call)
data(logreg.savefit5)
print(logreg.savefit5$call)
```

```
data(logreg.savefit6)
print(logreg.savefit6$call)
data(logreg.savefit7)
print(logreg.savefit7$call)
```

---

logreg.testdat   *Test data for Logic Regression*

Description

logreg.testdat has 500 cases, and 21 columns. Column 1 is the response Y, column k+1, k=1,...,20 is (binary) predictor Xk. Each predictor Xk is simulated as an independent Bernoulli(pk) random variables, with success probabilities pk between 0.1 and 0.9. The response variable is simulated from the model

Y = 3 + 1 L1 - 2 L2 + N(0,1),

where L1=(X1 or X2) and L2=(X3 or X4). So the task is to use the linear model in the logic regression framework to find L1 and L2.

See Also

[logreg](#)

Examples

```
data(logreg.testdat)
```

---

logreg.tree.control   *Control for logreg*

Description

Control of various secondary parameters of tree shape needed in `logreg`.

Usage

```
logreg.tree.control(treesize=8, opers=1, minmass=0, n1)
```

Arguments

treesize   specify the maximum number of allowed leaves per logic tree. Allowing one leave means that the tree is (at most) a simple predictor, two leaves allows for trees such as (X1 or X2) or (not X3 and X4). Four, eight and sixteen leaves allow for two, three or four levels of operators. To be able to interpret the results, do not choose too many leaves. Since the model selection techniques usually trim down the trees, it is recommend to allow at least four or eight leaves per tree.

opers	The default is to allow both "and" and "or" operators in the logic trees. If the interest is in logic statements in disjunctive normal form, use only one of the two operator types. Choose 1 for both operators, 2 for only "and" and 3 for only "or".
minmass	specify the minimum number of cases for which any tree needs to be 1 and for which any tree needs to be 0 to be considered as a logic tree in the model. This is to prevent that logreg, will select trees with, for example, 999 1s and one 0 out of 1000 cases. The default is to take 5% of the cases or 15, whatever is less.
n1	if you specify the sample size n1, it is checked that minmass is smaller than n1/4. This option is used by logreg, but is likely not useful for direct use.

Details

Missing arguments take defaults. If the argument treesize is a list with arguments treesize, opers, and minmass, those values take precedent of directly specified values.

Value

A list with components treesize, opers, and minmass, that can be used as the value of the argument tree.control of logreg.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Selected chapters from the dissertation of Ingo Ruczinski, available from https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf

See Also

logreg, logreg.anneal.control, logreg.mc.control

Examples

```
mytreecontrol <- logreg.tree.control(treesize = 16, minmass = 10)
```

---

logregmodel   *Format of class logregmodel*

Description

This help file contains a description of the format of class logregmodel.

Usage

```
logregmodel()
```

Value

An object of class logregtree has the following components:

ntrees   the number of trees in the current model.

nleaves   the number of leaves for the fitted model.

coef   the coefficients for this model.

score   the score of the fitted model.

trees   a list of `ntrees` objects of class `logregtree`.

Author(s)

Ingo Ruczinski `<ingo@jhu.edu>` and Charles Kooperberg `<clk@fredhutch.org>`.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Kooperberg C, Ruczinski I, LeBlanc ML, Hsu L (2001). Sequence Analysis using Logic Regression, *Genetic Epidemiology*, **21**, S626-S631.

Selected chapters from the dissertation of Ingo Ruczinski, available from `https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf`

See Also

`logreg`, `plot.logregmodel`, `print.logregmodel`, `logregtree`

Examples

```
logregmodel()       # displays this help file
help(logregmodel)   # equivalent
```

---

logregtree   *Format of class logregtree*
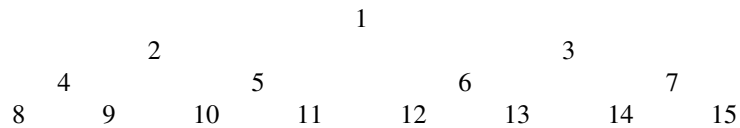

Description

This help file contains a description of the format of class logregtree.

Usage

```
logregtree()
```

Details

When storing trees, we number the location of the nodes using the following scheme (this is an example for a tree with at most 8 *terminal* nodes, but the generalization should be obvious):
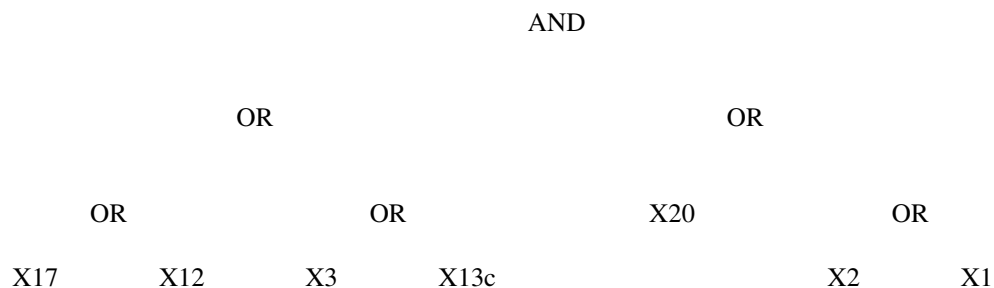
```
                          1
             2                        3
        4         5           6            7
      8    9    10    11    12    13    14    15
```

Each node may or may not be present in the current tree. If it is present, it can contain an operator ("and" or "or"), in which case it has to child nodes, or it can contain a variable, in which case the node is a terminal node. It is also possible that the node does not exist (as the user only specifies the maximum tree size, not the tree size that is actually fitted).

Output files have one line for each node. Each line contains 5 numbers:

1. the node number.

2. does this node contain an "and" (1), an "or" (2), a variable (3), or is the node empty (0).

3. if the node contains a variable, which one is it; e.g. if this number is 3 the node contains X3.

4. if the node contains a variable, does it contain the regular variable (0) or its complement (1)

5. is the node empty (0) or not (1) (this information is redundant with the second number)

Example

```
                                AND

             OR                              OR

      OR            OR               X20            OR

  X17     X12     X3     X13c                    X2     X1
```

is represented as

1	1	0	0	1
2	2	0	0	1
3	2	0	0	1
4	2	0	0	1
5	2	0	0	1
6	3	20	0	1
7	2	0	0	1
8	3	17	0	1
9	3	12	0	1
10	3	3	0	1
11	3	13	1	1
12	0	0	0	0
13	0	0	0	0
14	3	2	0	1
15	3	1	0	1

Value

An object of class logregtree is typically a substructure of an object of the class `logregmodel`. It will typically be the result of using the fitting function `logreg`. An object of class logictree has the following components:

whichtree	the sequence number of the current tree within the model.
coef	the coefficients of this tree.
trees	a matrix (data.frame) with five columns; see below for the format.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Selected chapters from the dissertation of Ingo Ruczinski, available from https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf

See Also

logreg, plot.logregtree, print.logregtree, logregmodel

Examples

```
logregtree()       # displays this help file
help(logregtree)   # equivalent
```

---

plot.logreg   *Plots for Logic Regression*

Description

Makes plots for objects fitted by `logreg`.

Usage

```
## S3 method for class 'logreg'
plot(x, pscript=FALSE, title=TRUE, ...)
```

Arguments

x   object of class `logreg`, typically the result of the function `logreg`.

pscript   if `TRUE` all plots will be stored in postscript files with distinct names.

title   if `TRUE` this generates a title for some plots, typically listing the number of trees and the model size.

...   graphical parameters can be given as arguments to plot.

Value

The type of the plots generated depends on the value of `x$select`.

if `select = 1` the fitted trees for the results of *find the best scoring model of any size* are plotted;

if `select = 2` or `select = 6` the fitted trees are plotted, as well as a graph of the scores for various model sizes versus model size for the results of *find the best scoring models for various sizes*, or *fit a sequence of logic regression models using a stepwise greedy algorithm*;

if `select = 3` training and test set scores as a function of model size for the results of *carry out cross-validation for model selection* are plotted;

if `select = 4` a histogram of the permutation scores with various important values highlighted for the results of *carry out a permutation test to check for signal in the data* are plotted;

if `select = 5` a series of histograms of the permutation scores with various important values highlighted for the results of *carry out a permutation test for model selection* are plotted;

if `select = 7` a histogram of the size frequency of the fitted models, a histogram of the frequency that predictors are marginally in the model, and (if that information was collected) an image plot for the observed frequency that predictors were jointly in the model, and an image plot of the observed/expected ratio of that joint frequency. See Kooperberg and Ruczinski (2004) for the definition of the expected frequency.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Kooperberg C, Ruczinki I (2005). Identifying interacting SNPs using Monte Carlo Logic Regression, *Genetic Epidemiology*, **28**, 157-170.

Selected chapters from the dissertation of Ingo Ruczinski, available from https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf

See Also

logreg, plot.logregmodel, plot.logregtree, logreg.testdat

Examples

```
data(logreg.savefit1,logreg.savefit2,logreg.savefit3,logreg.savefit4,
     logreg.savefit5,logreg.savefit6,logreg.savefit7)
#
# fit a single model
# myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 1000)
# logreg.savefit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21],
#               type = 2, select = 1, ntrees = 2, anneal.control = myanneal)
# the best score should be in the 0.96-0.98 range
plot(logreg.savefit1)
#
# fit multiple models
# myanneal2 <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 0)
# logreg.savefit2 <- logreg(select = 2, ntrees = c(1,2), nleaves =c(1,7),
#               oldfit = logreg.savefit1, anneal.control = myanneal2)
plot(logreg.savefit2)
# After an initial steep decline, the scores only get slightly better
# for models with more than four leaves and two trees.
#
# cross validation
# logreg.savefit3 <- logreg(select = 3, oldfit = logreg.savefit2)
plot(logreg.savefit3)
# 4 leaves, 2 trees should give the best test set score
#
# null model test
# logreg.savefit4 <- logreg(select = 4, anneal.control = myanneal2, oldfit = logreg.savefit1)
plot(logreg.savefit4)
# A histogram of the 25 scores obtained from the permutation test. Also shown
# are the scores for the best scoring model with one logic tree, and the null
# model (no tree). As the permutation scores are not even close to the score
```

```
# of the best model with one tree (fit on the original data), there is strong
# evidence against the null hypothesis that there was no signal in the data.
#
# Permutation tests
# logreg.savefit5 <- logreg(select = 5, oldfit = logreg.savefit2)
plot(logreg.savefit5)
# The permutation scores improve until we condition on a model with two
# trees and four leaves, and then do not change very much anymore. This
# indicates that the best model has indeed four leaves.
#
# a greedy sequence
# logreg.savefit6 <- logreg(select = 6, ntrees = 2, nleaves =c(1,12), oldfit = logreg.savefit1)
plot(logreg.savefit6)
# Monte Carlo Logic Regression
# logreg.savefit7 <- logreg(select = 7, oldfit = logreg.savefit1, mc.control=
#                    logreg.mc.control(nburn=1000, niter=100000, hyperpars=log(2)))
plot(logreg.savefit7)
```

---

plot.logregmodel   *Plots for Logic Regression*

Description

Makes plots for an object of class logregmodel fitted by logreg.

Usage

```
## S3 method for class 'logregmodel'
plot(x, pscript=FALSE, title=TRUE, nms, ...)
```

Arguments

x	object of class logregmodel, typically a part of an object of class logreg, which is the result of the function logreg.
pscript	if TRUE all plots will be stored in postscript files with distinct names.
title	if TRUE this generates a title for some plots, typically listing the number of trees and the model size.
nms	names of variables. If nms is provided variable names will be plotted, otherwise indices will be used.
...	graphical parameters can be given as arguments to plot.

Value

The fitted trees are plotted.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Selected chapters from the dissertation of Ingo Ruczinski, available from `https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf`

See Also

`logreg`, `logregmodel`, `plot.logreg`, `logreg.testdat`

Examples

```
data(logreg.savefit1)
# myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 1000)
# logreg.savefit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21],
#                   type = 2, select = 1, ntrees = 2, anneal.control = myanneal)
# plot(logreg.savefit1)
plot(logreg.savefit1$model) # does the same
```

---

plot.logregtree   *A plot of one Logic Regression tree.*

Description

Makes a plot of one Logic Regression tree, fitted by `logreg`.

Usage

```
## S3 method for class 'logregtree'
plot(x, nms, full=TRUE, and.or.cx=1.0, leaf.sz=1.0,
                leaf.txt.cx=1.0, coef.cx=1.0, indents=rep(0,4), coef=TRUE,
                coef.rd=4, ...)
```

Arguments

x	an object of class `logregtree`, or the `trees` component of such an object. Typically this object will be part of the result of an object of class `logreg`, generated with `select` = 1 (single model fit) or `select` = 2 (multiple model fit).
nms	names of variables. If nms is provided variable names will be plotted, otherwise indices will be used.
full	if TRUE, the tree occupies the entire window with margins specified by `indents`.
and.or.cx	character expansion (size) for the operators and/or.
leaf.sz	character expansion for the size of the leaves.

`leaf.txt.cx`	character expansion for the text in the leaves.
`coef.cx`	character expansion for the coefficient string.
`indents`	indents for plot - bottom, left, top, right.
`coef`	if `TRUE`, the coefficient of the tree is plotted.
`coef.rd`	controls how many digits of the above coefficient are displayed.
`...`	graphical parameters can be given as arguments to plot.

Value

This function makes a plot of one logic tree. The character expansion terms (`and.or.cx`, `leaf.sz`, `leaf.txt.cx`, `coef.cx`) defaults of 1.0 are chosen to generate a pretty plot of a single tree with up to eight leaves (4 levels deep). To plot more than one tree, or trees of different complexity, scale accordingly.

Author(s)

Ingo Ruczinski `<ingo@jhu.edu>` and Charles Kooperberg `<clk@fredhutch.org>`.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

Selected chapters from the dissertation of Ingo Ruczinski, available from `https://research.fredhutch.org/content/dam/stripe/kooperberg/ingophd-logic.pdf`

See Also

`logreg`, `frame.logreg`, `logreg.testdat`

Examples

```
data(logreg.savefit2)
#
# myanneal2 <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 0)
# logreg.savefit2 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21],
#                 type = 2, select = 2, ntrees = c(1,2), nleaves =c(1,7),
#                 anneal.control = myanneal2)
for(i in 1:logreg.savefit2$nmodels) for(j in 1:logreg.savefit2$alltrees[[i]]$ntrees[1]){
   plot.logregtree(logreg.savefit2$alltrees[[i]]$trees[[j]])
   title(main=paste("model",i,"tree",j))
}
```

| predict.logreg | *Predicted values Logic Regression* |

Description

Computes predicted values for one or more Logic Regression models that were fitted by a single call to `logreg`.

Usage

```
## S3 method for class 'logreg'
predict(object, msz, ntr, newbin, newsep, ...)
```

Arguments

object	Object of class `logreg`, that resulted from applying the function `logreg` with `select = 1` (single model fit), `select = 2` (multiple model fit), or `select = 6` (greedy stepwise fit).
msz	if `predict.logreg` is executed on an object of class `logreg`, that resulted from applying the function `logreg` with `select = 2` (multiple model fit) or `select = 6` (greedy stepwise fit) all logic trees for all fitted models are returned. To restrict the model size and the number of trees to some models, specify `msz` and `ntr` (for `select = 2`) or just `msz` (for `select = 6`).
ntr	see `msz`
newbin	binary predictors to evaluate the logic trees at. If `newbin` is omitted, the original (training) data is used.
newsep	separate (linear) predictors. If `newbin` is omitted, the original (training) predictors are used, even if `newsep` is specified.
...	other options are ignored

Details

This function calls `frame.logreg`.

Value

If `object$select = 1`, a vector with fitted values, otherwise a data frame with fitted values, where columns correspond to models.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

See Also

logreg, frame.logreg, logreg.testdat

Examples

```
data(logreg.savefit1,logreg.savefit2,logreg.savefit6,logreg.testdat)
#
# myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 1000)
# logreg.savefit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21], type = 2,
#                 select = 1, ntrees = 2, anneal.control = myanneal)
z1 <- predict(logreg.savefit1)
plot(z1, logreg.testdat[,1]-z1, xlab="fitted values", ylab="residuals")
# myanneal2 <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 0)
# logreg.savefit2 <- logreg(select = 2, nleaves =c(1,7), oldfit = logreg.savefit1,
#                 anneal.control = myanneal2)
z2  <- predict(logreg.savefit2)
# logreg.savefit6 <- logreg(select = 6, ntrees = 2, nleaves =c(1,12), oldfit = logreg.savefit1)
z6 <- predict(logreg.savefit6, msz = 3:5)
```

| print.logreg | *Prints Logic Regression Output* |

Description

Prints formulas for objects fitted by `logreg`.

Usage

```
## S3 method for class 'logreg'
print(x, nms, notnms, pstyle, ...)
```

Arguments

x	object of class `logreg`, typically the result of the function `logreg`.
nms	names of variables. If `nms` is provided variable names will be printted, otherwise `x$binnames` will be used. If that does not exist indices will be used.
notnms	names of complements of the variables. If `notnms` is not provided "not" will be added before the variable names.

pstyle   parenthesis style. If pstyle = 1 (the default) rules are more compact than if
                    pstyle = 2.

...   other options are ignored

Value

If x$select equals 1 or 2 the fitted logic rule(s) are generated as a text string. Scores, and if
x$select equals 2 or 6 modelsizes, are also provided. If x$select equals 4 or 5 a summary of the
permutation test(s) is printed. If x$select equals 3 a summary of the cross validation is printed. If
x$select is equal to 7 an error message is generated.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and
Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M.
Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

See Also

[logreg](), [print.logregmodel](), [print.logregtree](), [logreg.testdat]()

Examples

```
data(logreg.savefit1,logreg.savefit2,logreg.savefit3,logreg.savefit4,
     logreg.savefit5,logreg.savefit6)
#
# fit a single model
# myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 1000)
# logreg.savefit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21],
#                  type = 2, select = 1, ntrees = 2, anneal.control = myanneal)
# the best score should be in the 0.96-0.98 range
print(logreg.savefit1)
#
# fit multiple models
# myanneal2 <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 0)
# logreg.savefit2 <- logreg(select = 2, ntrees = c(1,2), nleaves =c(1,7),
#                  oldfit = logreg.savefit1, anneal.control = myanneal2)
print(logreg.savefit2)
# After an initial steep decline, the scores only get slightly better
# for models with more than four leaves and two trees.
#
# cross validation
# logreg.savefit3 <- logreg(select = 3, oldfit = logreg.savefit2)
print(logreg.savefit3)
# 4 leaves, 2 trees should give the best test set score
```

```
#
# null model test
# logreg.savefit4 <- logreg(select = 4, anneal.control = myanneal2, oldfit = logreg.savefit1)
print(logreg.savefit4)
# A summary of the permutation test
#
# Permutation tests
# logreg.savefit5 <- logreg(select = 5, oldfit = logreg.savefit2)
print(logreg.savefit5)
# A table summarizing the permutation tests
#
# a greedy sequence
# logreg.savefit6 <- logreg(select = 6, ntrees = 2, nleaves =c(1,12), oldfit = logreg.savefit1)
print(logreg.savefit6)
```

---

print.logregmodel	*Prints Logic Regression Formula*

Description

Prints formulas for objects fitted by `logreg`.

Usage

```
## S3 method for class 'logregmodel'
print(x, nms, notnms, pstyle, ...)
```

Arguments

x	object of class `logregmodel`, typically a part of an object of class `logreg`, which is the result of the function `logreg`.
nms	names of variables. If `nms` is provided variable names will be printed, otherwise indices will be used.
notnms	names of complements of the variables. If `notnms` is not provided "not" will be added before the variable names.
pstyle	parenthesis style. If `pstyle = 1` (the default) rules are more compact than if `pstyle = 2`.
...	other options are ignored

Value

A text representation of the model will be printed.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

See Also

[logreg](#), [logregmodel](#), [print.logreg](#), [print.logregtree](#), [logreg.testdat](#)

Examples

```
data(logreg.savefit1)
#
# myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 1000)
# logreg.savefit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21],
#                 type = 2, select = 1, ntrees = 2, anneal.control = myanneal)
print(logreg.savefit1$model)
```

---

  print.logregtree   *Prints Logic Regression Formula*

Description

Prints formulas for objects fitted by `logreg`.

Usage

```
## S3 method for class 'logregtree'
print(x, nms, notnms, pstyle, ...)
```

Arguments

x	object of class `logregtree`, typically a part of an object of class `logreg`, which is the result of the function `logreg`, or a matrix with five columns (see `logregtree`).
nms	names of variables. If `nms` is provided variable names will be printed, otherwise indices will be used.
notnms	names of complements of the variables. If `notnms` is not provided "not" will be added before the variable names.
pstyle	parenthesis style. If `pstyle = 1` (the default) rules are more compact than if `pstyle = 2`.
...	other options are ignored

Value

A text representation of the tree will be printed.

Author(s)

Ingo Ruczinski <ingo@jhu.edu> and Charles Kooperberg <clk@fredhutch.org>.

References

Ruczinski I, Kooperberg C, LeBlanc ML (2003). Logic Regression, *Journal of Computational and Graphical Statistics*, **12**, 475-511.

Ruczinski I, Kooperberg C, LeBlanc ML (2002). Logic Regression - methods and software. *Proceedings of the MSRI workshop on Nonlinear Estimation and Classification* (Eds: D. Denison, M. Hansen, C. Holmes, B. Mallick, B. Yu), Springer: New York, 333-344.

See Also

logreg, logregtree, print.logreg, print.logregmodel, logreg.testdat

Examples

```
data(logreg.savefit1)
#
# myanneal <- logreg.anneal.control(start = -1, end = -4, iter = 25000, update = 1000)
# logreg.savefit1 <- logreg(resp = logreg.testdat[,1], bin=logreg.testdat[, 2:21],
#                   type = 2, select = 1, ntrees = 2, anneal.control = myanneal)
print(logreg.savefit1$model$trees[[1]])
```

Index