Package 'mvboxcox'

August 26, 2026

Title Bivariate Logistic Box-Cox Regression

Version 0.1.4

Description Fits bivariate logistic Box-Cox regression models for binary outcomes and positive continuous predictors. Transformation parameters are selected by cross-validated grid search with adaptive refinement and thin-plate spline smoothing. The package also provides prediction, empirical and sampling-weighted median effects, simulation tools, and sampling-weighted model fitting. The methodology extends the logistic Box-Cox approach of Xing et al. (2021) <doi:10.1002/cjs.11587>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Imports methods, stats, parallel, fields

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Shiyu Xu [aut],
Xuekui Zhang [aut, cre]

Maintainer Xuekui Zhang <ubcxzhang@gmail.com>

Repository CRAN

Date/Publication 2026-08-26 20:10:02 UTC

Contents

`depress` *NHANES Depression and Blood Metals Data*

Description

A sampling-weighted analytic dataset derived from the 2005-2006 and 2007-2008 cycles of the US National Health and Nutrition Examination Survey (NHANES). The data are used in the bivariate logistic Box-Cox application described by Xu and Zhang.

Usage

```
depress
```

Format

A data frame with 8,893 rows and 6 variables:

depression Integer indicator: 1 for a Patient Health Questionnaire-9 (PHQ-9) score of at least 10, and 0 otherwise.

mercury Total blood mercury concentration in micrograms per liter (mcg/L).

blood_lead Blood lead concentration in micrograms per deciliter (mcg/dL).

age Age in years.

gender Integer indicator: 1 for male and 0 for female.

weight Four-year Day 1 dietary sampling weight, constructed for the combined NHANES cycles.

Details

The analytic sample contains adults aged 20 years or older with a valid Day 1 dietary sampling weight, complete PHQ-9 information, and nonmissing positive blood mercury and blood lead measurements. The package's survey-weighted implementation uses `weight` but does not incorporate NHANES strata or primary sampling-unit identifiers, so analyses should be interpreted as sampling-weighted rather than as complete design-based survey analyses.

Source

National Center for Health Statistics, National Health and Nutrition Examination Survey, 2005-2006 and 2007-2008 cycles.

References

Xu, S., & Zhang, X. *Bivariate logistic Box-Cox regression for interpretable nonlinear exposure-response modeling*. Manuscript.

Xing, L., Zhang, X., Burstyn, I., & Gustafson, P. (2021). On logistic Box-Cox regression for flexibly estimating the shape and strength of exposure-disease relationships. *Canadian Journal of Statistics*, 49(3), 808-825.

Examples

```
data(depress)
str(depress)
summary(depress)
```

`ld` *Boxcox transformation*

Description

`if (li == 0) log(xi) else (xi**li - 1)/li`

Usage

```
ld(xi, li)
```

Arguments

`xi`	positive numeric values to be Box-Cox transformed
`li`	i-th lambda value

Value

numeric vector of Box-Cox transformed values

mvbc.fullformula *Build the full Box-Cox model formula*

Description

Apply a set of lambda exponents to the positive predictor terms of `formulaA` (via `ld()`) and combine with the untransformed terms of `formulaB` into a single formula suitable for `glm()`.

Usage

```
mvbc.fullformula(formulaA, formulaB, lambdas)
```

Arguments

`formulaA`	a formula containing the binary outcome and the positive continuous predictors to be Box-Cox transformed
`formulaB`	an optional one-sided formula containing additional covariates that remain untransformed, or `NULL`
`lambdas`	vector of lambda exponents, one per term in `formulaA`, in the same order

Value

a `formula` object with the positive predictor terms of `formulaA` replaced by `ld(term, lambda)` and the terms of `formulaB` appended

mvbc.glm.step *Single train/test glm fit at a fixed lambda*

Description

Fits `glm()` to `train` at a given set of lambda exponents (weighted, if `survey = TRUE` and `weights` is supplied) and, if `test` is supplied, scores the fit on `test` and attaches `sspr`/`ssdr` (see `mvbc.trainer.ssr()`) to the returned model object.

Usage

```
mvbc.glm.step(
  formulaA,
  formulaB = NULL,
  train,
  test = NULL,
  lambdas,
  weights = NULL,
  test_weights = NULL,
  survey = FALSE
)
```

Arguments

formulaA	a formula containing the binary outcome and the positive continuous predictors to be Box-Cox transformed
formulaB	an optional one-sided formula containing additional covariates that remain untransformed, or NULL
train	training data frame
test	optional test data frame to score the fit on
lambdas	vector of lambda exponents, one per term in formulaA, in the same order
weights	optional vector of observation weights aligned with the rows of train
test_weights	optional vector of observation weights aligned with the rows of test, used when computing sspr/ssdr
survey	logical; if TRUE and weights is supplied, fit a weighted glm() instead of an unweighted glm()

Value

the fitted glm object, with sspr and ssdr attached when test is supplied

---

mvbc.median.effect *Median effects from a fitted BLBC model*

Description

Computes coefficient-like median effects for the positive predictors in a model fitted by mvbc.train(). The calculation uses the empirical median, or the sampling-weighted empirical median when weights is supplied, and therefore does not require a log-normal distribution.

Usage

```
mvbc.median.effect(object, data, q = 1, weights = NULL, na.rm = TRUE)
```

Arguments

object	a fitted mvbc.train.model returned by mvbc.train()
data	data frame containing the positive predictors used in object$formulaA
q	numeric reference-transformation value; either a scalar or one value per transformed predictor
weights	optional nonnegative sampling-weight vector aligned with the rows of data
na.rm	logical; if TRUE, observations with missing predictor values are omitted when calculating each median

Details

For predictor X_j and reference transformation q, the median effect is

$$\delta_j(q) = \beta_j m_j^{\lambda_j - q},$$

where m_j is the marginal median of X_j. The plug-in estimator replaces β_j, λ_j, and m_j with their fitted or empirical counterparts. If X_j is log-normal with location parameter μ_j, then $m_j = \exp(\mu_j)$ and the expression reduces to $\beta_j \exp\{(\lambda_j - q)\mu_j\}$.

With `weights`, medians are calculated as weighted empirical 0.5 quantiles: values are sorted ascending and the median is the value at which the weighted empirical CDF first reaches 0.5 (equivalent to a one-stage survey design with `ids = ~1`). The returned effects are on the chosen reference-transformation scale. Comparisons between predictors remain dependent on their measurement units.

Value

A data frame with one row per transformed predictor and columns `predictor`, `beta`, `lambda`, `q`, `median`, and `median_effect`.

Examples

```
fit <- structure(
  list(
    formulaA = y ~ x1 + x2,
    lambda.fits = data.frame(x1 = 0.5, x2 = 1),
    beta.fits = data.frame(
      foldid = 0L, intercept = -1, x1 = 2, x2 = -0.5
    )
  ),
  class = "mvbc.train.model"
)
dat <- data.frame(
  y = c(0, 0, 1, 1, 1),
  x1 = c(1, 2, 3, 7, 12),
  x2 = c(2, 3, 4, 5, 8)
)
mvbc.median.effect(fit, dat, q = 1)
```

---

mvbc.optimizer *Optimizer*

Description

The Multivariate Boxcox Optimizer accepts an instance of an `mvbc.trainer.model` and uses Thin Plate Splines (TPS) to search for more optimal lambda estimates.

Usage

```
mvbc.optimizer(
  model,
  data,
  tpsargs = list(scale.type = "unscaled"),
  optimargs = list(method = "L-BFGS-B", lower = 0, upper = 2),
  weights = NULL,
  survey = FALSE
)
```

Arguments

model	an instance of an mvbc.trainer.model.
data	a data frame containing the variables in the model
tpsargs	list of optional arguments to pass to Tps().
optimargs	list of optional arguments to pass to optim().
weights	optional vector of observation weights (e.g. survey weights) aligned with the rows of data, passed through to mvbc.trainer.fit()
survey	logical; if TRUE and weights is supplied, fit each candidate model with a weighted glm() instead of an unweighted glm()

Details

The mvbc.optimizer() function performs the following:

1. Starting with an instance of an mvbc.trainer.model repeat the following steps for each minima value in the instance.
2. Generate a Thin Plate Spline (TPS) surface using the lambda grid as the independent x variables in the TPS, and the some-of-square residual minima as the dependent y variable.
3. Use the general purpose optim() function to choose updated estimates of lambda to minimize each of the some-of-square residuals.
4. The updated lambda estimates are used to fit another set of beta values and compute some-of-square residuals as is done in the original mvbc.trainer().
5. Append the updated lambda and beta estimates and residuals to the existing training results. The updated estimates may or may not have improved residuals.

Value

mvbc.optimizer() returns an updated instance of the class mvbc.trainer.model as defined in the original mvbc.trainer().

Examples

```
vBeta = c(-2.2,-0.4,-0.2,-0.005)
vLambda = c(0.5,1.5)
vMean =  c(-0.08, -0.01, 50)
vSd =    c(0.932543,    0.8,       18.11569)
```

```
vNames = c("mercury",   "lead", "age")
sim = mvbc.simulator(vLambda, vBeta, vMean, vSd, vNames, 500, 1)
summary(sim)

# ignore "glm.fit: fitted probabilities numerically 0 or 1 occurred"
suppressWarnings({
  trained = mvbc.trainer(
    Ybin ~ mercury + lead, ~ age, sim$data,
    griddomain = seq(0, 2, length.out = 4), K = 3
  )
})
trained = mvbc.optimizer(trained, sim$data)

summary(trained)
```

`mvbc.predict` *Predict probability from a trained* `mvbc.train.model`

Description

Predict $\hat{p}$ on test data using a model fitted by `mvbc.train()`, i.e. one whose `lambda.fits` and `beta.fits` have already been resolved to a single chosen lambda tuple (unlike `mvbc.trainer.predict.p_hat()`, which predicts from a specific `tupleid` inside an `mvbc.trainer.model` that may hold many candidate tuples).

Usage

```
mvbc.predict(trainModel, data)
```

Arguments

`trainModel`	instance of the class `mvbc.train.model`, as returned by `mvbc.train()`
`data`	a data frame containing the test data $\mathbf{X}$ to predict on

Details

Given the definition of $\mathbf{Z}(\tilde{\lambda}, \tilde{\beta}, \mathbf{X})$ and $\hat{p}(\mathbf{Z}) = \mathbf{1}/(\mathbf{1}+\mathbf{e}^{-\mathbf{Z}})$ as shown `mvbc.simulator()`, `mvbc.predict()`:

1. Builds the full formula using the chosen $\hat{\lambda}$ in `trainModel$lambda.fits`.
2. Builds a model matrix from `data` using that formula.
3. Uses the full-data estimate $\hat{\beta}$ in `trainModel$beta.fits` to compute $\mathbf{Z}(\hat{\lambda}, \hat{\beta}, \mathbf{X})$ and $\hat{p}(\mathbf{X})$.
4. Returns the estimated probabilities.

Value

`mvbc.predict()` returns a vector of estimated probabilities, one per row of `data`.

Examples

```
vBeta = c(-2.2,-0.4,-0.2,-0.005)
vLambda = c(0.5,1.5)
vMean =  c(-0.08, -0.01, 50)
vSd =    c(0.932543,    0.8,       18.11569)
vNames = c("mercury",   "lead", "age")
trainset = mvbc.simulator(vLambda, vBeta, vMean, vSd, vNames, 500, 1)
testset = mvbc.simulator(simModel = trainset)

suppressWarnings({
  trained = mvbc.train(
    Ybin ~ mercury + lead, ~ age, trainset$data,
    griddomain = seq(0, 2, length.out = 4), K = 3
  )
})
p_hat = mvbc.predict(trained, testset$data)
head(p_hat)
```

`mvbc.simulator` *Simulator*

Description

The Multivariate Boxcox Simulator `mvbc.simulator` generates binomial samples from a logistic model built from both Boxcox transformed log normal random variables and normal random variables. The simulator is given a set of model parameters, and returns a class object with the parameters, the random variables, the logistic output $\mathbf{Z}$ and the binomial samples $\mathbf{Y_{bin}}$. The simulator includes controlled seeding to allow for generation of the same sequence of simulated data each time.

Also read the introduction.

Usage

```
mvbc.simulator(vLambda, vBeta, vMean, vSd, vNames, n, seed, simModel = NULL)
```

Arguments

`vLambda`	vector of Boxcox exponents $\vec{\lambda}$ for the log normal random variables. The length of this vector determines the number c of log normal random variables.
`vBeta`	vector of the logistic model coefficients $\vec{\beta}$. The length of this vector determines the number k of normal random variables.
`vMean`	vector of the means of the random variables $\vec{\mu}$
`vSd`	vector of the standard deviations of the random variables $\vec{\sigma}$
`vNames`	vector of names for the random variables
`n`	integer number of samples to generate

seed	integer value of initial random seed
simModel	instance of the class mvbc.simulator.model used to generate additional data from existing settings

Details

The basic Multivariate Boxcox logistic model is expressed as:

$$\mathbf{Z}(\tilde{\lambda}, \tilde{\beta}, \mathbf{X}) = \beta_{\mathbf{0}} + \beta_{\mathbf{1}}\mathbf{X_1}^{(\lambda_1)} + ... + \beta_{\mathbf{c}}\mathbf{X_c}^{(\lambda_c)} + \beta_{\mathbf{c+1}}\mathbf{X_{c+1}} + ... + \beta_{\mathbf{c+k}}\mathbf{X_{c+k}}$$

where the probability of a positive event $p = E(y = 1|\mathbf{X})$ is estimated as:

$$\hat{\mathbf{p}}(\mathbf{Z}) = \mathbf{1}/(\mathbf{1} + \mathbf{e}^{-\mathbf{Z}})$$

and the Boxcox transformation $x^{(\lambda)}$ is:

- $(x^\lambda - 1)/\lambda$, if $\lambda \neq 0$
- $ln(x)$, if $\lambda = 0$

The contributing variables are:

- $\mathbf{X_a}$ are the log normal random variables where $1 < a < c$
- $\mathbf{X_b}$ are the normal random variables where $c + 1 < b < c + k$
- c is the number of log normal random variables
- k is the number of normal random variables
- $\vec{\beta}$ is a vector of the logistic model coefficients
- $\vec{\mu}$ is a vector of the means of the random variables $\mathbf{X_a}, \mathbf{X_b}$
- $\vec{\sigma}$ is a vector of the standard deviations of the random variables $\mathbf{X_a}, \mathbf{X_b}$
- $\vec{\lambda}$ is a vector of Boxcox exponents for the log normal random variables $\mathbf{X_a}$

The simulator will generate n samples for $\mathbf{X_a}$ and $\mathbf{X_b}$ and then compute the vector $\mathbf{Z}$. The samples are generated using the R functions $rlnorm(n, \mu, \sigma)$ and $rnorm(n, \mu, \sigma)$ respectively. The vector $\mathbf{Z}$ is then used to generate n binomial samples $\mathbf{Y_{bin}}$ using the R function:

$$\mathbf{Y_{bin}} = \mathbf{rbinom}(\mathbf{n}, \mathbf{1}, \mathbf{1}/(\mathbf{1} + \mathbf{e}^{-\mathbf{Z}}))$$

An initial first usage call to mvbc.simulator() will set the given seed using set.seed(). Subsequent calls using the object will increment the seed first, and then call set.seed().

Value

mvbc.simulator() returns an object of class mvbc.simulator.model which contains the following components:

vLambda	vector of Boxcox exponents
vBeta	vector of the logistic model coefficients
vMean	vector of the means of the random variables

vSd	vector of the standard deviations of the random variables
vNames	vector of names for the random variables
n	integer number of samples to generate
c	integer number of log normal, Boxcox transformed variables
k	integer number of normal variables
seed	integer value of the last used random seed
ssr	sum or square residuals with self
cm	caret confusion matrix result with self
data	matrix with n rows and $c + k + 2$ columns. The first $c + k$ columns are the samples of the random variables $\mathbf{X_i}$, named using the vNames vector. The last two columns are the $\mathbf{Z}$ and $\mathbf{Y_{bin}}$ results.

Examples

```
vBeta = c(-2.2,-0.4,-0.2,-0.005)
vLambda = c(0.5,1.5)
vMean =  c(-0.08, -0.01, 50)
vSd =    c(0.932543,    0.8,       18.11569)
vNames = c("mercury",   "lead", "age")
sim = mvbc.simulator(vLambda, vBeta, vMean, vSd, vNames, 1000, 1)
summary(sim)
```

---

mvbc.tester *Tester*

Description

The Multivariate Boxcox Tester provides a means to predict binary outcomes for a test data set, given a trained model from the Multivariate Boxcox Trainer Core. The Tester can also run multiple train and test cycles using the Multivariate Boxcox Simulator to generate matching training data and test data. A variety of evaluation metrics are available including the Pearson and Deviance residuals and the F1-score and related metrics.

Also read the introduction.

Usage

```
mvbc.tester(
  formulaA,
  formulaB = NULL,
  simModel,
  nTrain,
  nTest,
  griddomain = seq(0, 2, l = 5),
  K = 5,
```

```
  depth = 2,
  cores = 1,
  minima = c("ssdr"),
  tpsargs = list(scale.type = "unscaled"),
  optimargs = list(method = "L-BFGS-B", lower = 0, upper = 2)
)
```

Arguments

formulaA	a formula containing the binary outcome and the positive continuous predictors to be Box-Cox transformed
formulaB	an optional one-sided formula containing additional covariates that remain untransformed
simModel	instance of the class mvbc.simulator.model provides bases for generating additional training and test sets
nTrain	number of training sets to generate
nTest	number of test sets to generate
griddomain	vector or list of vectors defining the domain of candidate lambda values. If it is a list, then must contain c elements, one vector for each lambda in the model.
K	number of folds for cross validation
depth	allows for recursion to finer grained lambda grids. A value of depth = 1 means no recursion.
cores	the number of cores for parallel processing
minima	which minima to use for deeper grid and spline in the trainer, defaults to c("ssdr")
tpsargs	list of optional arguments to pass to Tps().
optimargs	list of optional arguments to pass to optim().

Details

The mvbc.tester() function performs the following:

1. Using simModel as a basis, call mvbc.simulator() to generate nTrain training data sets and nTest test data sets. Relies on the embedded seed to generate repeatable results.
2. For each of the nTrain training data sets, fit an mvbc.trainer.model mvbc.trainer() and then optimize the fit using mvbc.optimizer().
3. For the given minima:
   (a) For each nTest test data set, estimate $\hat{p}$ using mvbc.trainer.predict.p_hat() then estimate $\hat{\mathbf{Y}}_{bin}$. Compute sum of square Pearson and Deviance residuals, and a confusion matrix report cm. Save results in an instance of a mvbc.tester.result
4. Return an instance of mvbc.tester.result.

When estimating $\hat{\mathbf{Y}}_{bin}$ for use in the confusion matrix, the trainModel$prior will be used to define the cut-off for $\hat{p}$.

Value

`mvbc.tester()` returns an object of class `mvbc.tester.result` which contains the following components:

`nTrain`	number of training sets to generate
`nTest`	number of test sets to generate
`griddomain`	list of c vectors defining the domain of candidate lambda values
`formulaA`	a formula containing the binary outcome and positive continuous predictors to be Box-Cox transformed.
`formulaB`	an optional one-sided formula containing additional covariates that remain untransformed.
`simModel`	instance of last returned `mvbc.simulator.model`
`K`	number of folds for cross validation
`depth`	allows for recursion to finer grained lambda grids. A value of `depth = 1` means no recursion.
`minima`	name of minima selector(s) were used for finer grained grid and Thin Plate Spline optimization
`lambda.fits`	data frame containing estimated lambdas, residuals, F1 scores. Full metrics are in the `cms` list.
`beta.fits`	data frame containing estimated betas.
`cms`	list of caret confusion matrix results, corresponding to each row in `labda.fits`

Examples

```
vBeta = c(-2.2,-0.4,-0.2,-0.005)
vLambda = c(0.45,1.50)
vMean =  c(-0.08, -0.01, 50)
vSd =    c(0.932543,    0.8,       18.11569)
vNames = c("mercury",   "lead", "age")
sim = mvbc.simulator(vLambda, vBeta, vMean, vSd, vNames, 300, 1)

# one training sample and one test sample
# ignore "glm.fit: fitted probabilities numerically 0 or 1 occurred"
suppressWarnings({
    test.result = mvbc.tester(
      Ybin ~ mercury + lead, ~ age, sim,
      nTrain = 1, nTest = 1,
      griddomain = seq(0, 2, length.out = 3),
      K = 2, depth = 1
    )
})
summary(test.result)

maxF1 = which.max(test.result$lambda.fits$F1)
test.result$cms[maxF1]
```

mvbc.train *Train*

Description

Full pipeline: runs `mvbc.trainer()` to grid-search candidate lambdas, then `mvbc.optimizer()` to refine the selected minima with a Thin Plate Spline, refits the regression coefficients on the full data set at the selected lambda, and returns a fitted `mvbc.train.model` with `lambda.fits` and `beta.fits` populated at the chosen `minima`.

Usage

```
mvbc.train(
  formulaA,
  formulaB = NULL,
  data,
  griddomain = seq(0, 2, l = 10),
  K = 5,
  depth = 1,
  cores = 1,
  minima = c("ssdr"),
  tpsargs = list(scale.type = "unscaled"),
  optimargs = list(method = "L-BFGS-B", lower = 0, upper = 2),
  seed = 111,
  weights = NULL,
  survey = FALSE
)
```

Arguments

`formulaA`	a formula containing the binary outcome and the positive continuous predictors to be Box-Cox transformed
`formulaB`	an optional one-sided formula containing additional covariates that remain untransformed
`data`	a data frame containing the variables in the model
`griddomain`	vector or list of vectors defining the domain of candidate lambda values. If it is a list, then must contain c elements, one vector for each lambda in the model.
`K`	number of folds for cross validation
`depth`	allows for recursion to finer grained lambda grids. A value of `depth = 1` means no recursion.
`cores`	the number of cores for parallel processing
`minima`	a single minima name (e.g. `"ssdr"`) used both for the deeper grid/spline refinement and to select the one final lambda/beta refit `mvbc.train()` returns; unlike `mvbc.trainer()` and `mvbc.tester()`, `mvbc.train()` cannot report more than one final model, so `length(minima)` must be 1

tpsargs	list of optional arguments to pass to Tps().
optimargs	list of optional arguments to pass to optim().
seed	random seed used to assign cross-validation folds
weights	optional vector of observation weights (e.g. survey weights), passed through to mvbc.trainer() and mvbc.optimizer()
survey	logical; if TRUE and weights is supplied, fit each candidate model with a weighted glm() instead of an unweighted glm()

Value

mvbc.train() returns an object of class mvbc.train.model, which extends mvbc.trainer.model (see mvbc.trainer()) with lambda.fits and beta.fits set to the estimates at the chosen minima. beta.fits contains the single full-data refit, identified by foldid = 0.

Examples

```
vBeta = c(-2.2,-0.4,-0.2,-0.005)
vLambda = c(0.5,1.5)
vMean =  c(-0.08, -0.01, 50)
vSd =    c(0.932543,    0.8,       18.11569)
vNames = c("mercury",   "lead", "age")
sim = mvbc.simulator(vLambda, vBeta, vMean, vSd, vNames, 500, 1)

# ignore "glm.fit: fitted probabilities numerically 0 or 1 occurred"
suppressWarnings({
  trained = mvbc.train(
    Ybin ~ mercury + lead, ~ age, sim$data,
    griddomain = seq(0, 2, length.out = 4), K = 3, depth = 1
  )
})
summary(trained)
```

---

mvbc.trainer *Trainer*

Description

The Multivariate Box-Cox Trainer Core mvbc.trainer accepts positive continuous predictors $\mathbf{X_a}$ to be Box-Cox transformed, optional untransformed covariates $\mathbf{X_b}$, and a binary outcome $\mathbf{Y_{bin}}$. Using K-fold cross-validation, the trainer estimates logistic models for $\mathbf{Z}(\hat{\lambda}, \bar{\beta}, \mathbf{X})$ consisting of the estimated Box-Cox exponents $\hat{\lambda}$ and K sets of estimated coefficients $\hat{\beta}$.

Also read the introduction.

Usage

```
mvbc.trainer(
  formulaA,
  formulaB = NULL,
  data,
  griddomain = seq(0, 2, l = 10),
  K = 5,
  depth = 1,
  cores = 1,
  minima = c("ssdr"),
  seed = 111,
  weights = NULL,
  survey = FALSE
)
```

Arguments

formulaA	a formula containing the binary outcome and the positive continuous predictors to be Box-Cox transformed
formulaB	an optional one-sided formula containing additional covariates that remain untransformed
data	a data frame containing the variables in the model
griddomain	vector or list of vectors defining the domain of candidate lambda values. If it is a list, then must contain c elements, one vector for each lambda in the model.
K	number of folds for cross validation|
depth	allows for recursion to finer grained lambda grids. A value of depth = 1 means no recursion.
cores	the number of cores for parallel processing
minima	which minima or minimas to use for deeper grid and spline, defaults to c("ssdr")
seed	random seed used to assign cross-validation folds
weights	optional vector of observation weights (e.g. survey weights) aligned with the rows of data, passed through to mvbc.trainer.fit() and mvbc.glm.step()
survey	logical; if TRUE and weights is supplied, fit each candidate model with a weighted glm() instead of an unweighted glm()

Details

Recall from mvbc.simulator() that the basic Multivariate Boxcox logistic model is expressed as:

$$\mathbf{Z}(\tilde{\lambda}, \tilde{\beta}, \mathbf{X}) = \beta_{\mathbf{0}} + \beta_{\mathbf{1}}\mathbf{X_1}^{(\lambda_1)} + ... + \beta_{\mathbf{c}}\mathbf{X_c}^{(\lambda_c)} + \beta_{\mathbf{c+1}}\mathbf{X_{c+1}} + ... + \beta_{\mathbf{c+k}}\mathbf{X_{c+k}}$$

where:

- c is the number of positive predictors to be transformed
- k is the number of additional untransformed covariates

- $x^{(\lambda)}$ is the Box-Cox transformation
- $\hat{p}(\mathbf{Z}) = \mathbf{1}/(\mathbf{1} + \mathbf{e}^{-\mathbf{Z}})$ is the estimated probability of a positive event

The `mvbc.trainer()` algorithm is summarized here:

1. Creates a Lambda `grid` matrix from the `griddomain`. The `grid` matrix contains all the possible candidate tuples of Lambda, with additional columns `sspr` and `ssdr` to contain sum-of-squares of Pearson and Deviance residuals. For example, if each lambda domain contains 10 values, then there would be $G = 10^c$ tuples.
2. Split the `data` into K-folds (reuse folds if available in `object$folds`).
3. Create a `gridfits` matrix where each row will store `tupleid`, `foldid`, and the $\hat{\beta}$ estimates from a `glm()` model.
4. For each fold, create training-set and test-set...
   (a) For each lambda tupleid in `grid`...
      i. Fit $\mathbf{Z}(\tilde{\lambda}, \tilde{\beta}, \mathbf{X})$ to the training-set using `glm()`.
      ii. Use the glm model to estimate probabilities $\hat{\mathbf{p}}$ from the test-set.
      iii. Use $\mathbf{Y_{bin}}$ and $\hat{\mathbf{p}}$ from the test-set to compute Pearson and Deviance residuals and accumulate in `grid`.
      iv. Save the glm model coefficients as the $\hat{\beta}$ estimates in the `gridfits` matrix.
5. Find lambda `tupleids` for the minimum `sspr` and `ssdr` and save in `minimas` list.
6. While `depth` > 1 then:
   (a) Decrement `depth`.
   (b) Compute a finer `griddomain` centered on chosen minima. Repeat steps 3 to 5 and merge the the `grid`, `gridfits`, and `minimas` results.

Value

`mvbc.trainer()` returns an object of class `mvbc.trainer.model` which contains the following components:

`prior`	estimated prior probability $E(y = 1) = \frac{1}{n} \sum (Y_{bin} = 1)$
`grid`	$G \times (c + 2)$ matrix of lambda tuples. G is the number of tuples and c is the number of positive predictors to be transformed. The additional two columns are `sspr` and `ssdr` are the sum-of-squares of the Pearson and Deviance residuals.
`griddomain`	list of c vectors defining the domain of candidate lambda values
`gridfits`	$(G * K) \times (c + k + 3)$ matrix of lambda `tupleid`, `foldid`, and $\hat{\beta}$ estimates from each K-fold cross validation of each lambda tuple. The `tupleid` refers to a row in `grid`.
`minimas`	list of minima selectors of best lambdas of the form `list(ssdr=tupleid1, sspr=tupleid2)`
`minima`	name of minima selector(s) were used for finer grained grid and Thin Plate Spline optimization
`folds`	vector of `foldids`, one entry for each row in `data`
`formulaA`	a formula containing the binary outcome and positive continuous predictors to be Box-Cox transformed.

formulaB	an optional one-sided formula containing additional covariates that remain untransformed.
K	number of folds for cross validation
depth	allows for recursion to finer grained lambda grids. A value of depth = 1 means no recursion.

Examples

```
vBeta = c(-2.2,-0.4,-0.2,-0.005)
vLambda = c(0.5,1.5)
vMean =  c(-0.08, -0.01, 50)
vSd =    c(0.932543,    0.8,       18.11569)
vNames = c("mercury",   "lead", "age")
sim = mvbc.simulator(vLambda, vBeta, vMean, vSd, vNames, 500, 1)
summary(sim)

# ignore "glm.fit: fitted probabilities numerically 0 or 1 occurred"
suppressWarnings({
  trained = mvbc.trainer(
    Ybin ~ mercury + lead, ~ age, sim$data,
    griddomain = seq(0, 2, length.out = 4), K = 3
  )
})
summary(trained)
```

---

mvbc.trainer.fit *Workhorse to fit instance of* mvbc.trainer.model

---

Description

Workhorse to fit instance of mvbc.trainer.model

Usage

```
mvbc.trainer.fit(
  trainModel,
  data,
  gridid.start = 1,
  weights = NULL,
  survey = FALSE
)
```

Arguments

trainModel	instance of the class mvbc.trainer.model used for finer grain recursion.
data	a data frame containing the variables in the model
gridid.start	tuple grid id to begin fitting

`weights`	optional vector of observation weights (e.g. survey weights) aligned with the rows of `data`, split into train/test folds and passed to `mvbc.glm.step()`
`survey`	logical; if `TRUE` and `weights` is supplied, fit each candidate model with a weighted `glm()` instead of an unweighted `glm()`

Value

An updated object of class `mvbc.trainer.model`. Its `grid` component contains the accumulated cross-validation residual criteria for each lambda tuple, and `gridfits` contains the corresponding fold-specific regression coefficients. All other training metadata in `trainModel` is retained.

`mvbc.trainer.predict.p_hat`
Predict probability

Description

Predict $\hat{p}$ on test data using training model

Usage

```
mvbc.trainer.predict.p_hat(trainModel, data, tupleid)
```

Arguments

`trainModel`	instance of the class `mvbc.trainer.model`
`data`	a data frame containing the test data $\mathbf{X}$ to predict on
`tupleid`	which grid tuple from the `trainModel` to predict on

Details

Given the definition of $\mathbf{Z}(\tilde{\lambda}, \tilde{\beta}, \mathbf{X})$ and $\hat{p}(\mathbf{Z}) = \mathbf{1}/(\mathbf{1} + \mathbf{e}^{-\mathbf{Z}})$ as shown `mvbc.simulator()`, the `mvbc.trainer.predict.p_hat()` function performs the following:

1. Look up the estimated $\hat{\lambda}$ values associated with the given `tupleid`.
2. Look up the K sets of estimated $\hat{\beta}$ values associated with the given `tupleid`.
3. For each of the K sets of $\hat{\beta}$ values, compute an estimated value of $\mathbf{Z}(\hat{\lambda},\hat{\beta}, \mathbf{X})$ and $\hat{p}(\mathbf{X})$ from the test `data`
4. Compute the mean of the K estimates of $\hat{p}(\mathbf{Z})$ as $\hat{p}_{mean}(\mathbf{Z})$ and return this vector of mean $\hat{p}$ values.

Value

`mvbc.trainer.predict.p_hat()` returns vector of $\hat{p_{mean}}(\mathbf{Z})$ estimates.

Examples

```
vBeta = c(-2.2,-0.4,-0.2,-0.005)
vLambda = c(0.5,1.5)
vMean =  c(-0.08, -0.01, 50)
vSd =    c(0.932543,    0.8,       18.11569)
vNames = c("mercury",   "lead", "age")
trainset = mvbc.simulator(vLambda, vBeta, vMean, vSd, vNames, 500, 1)
testset = mvbc.simulator(simModel = trainset)

# train and predict one pair
suppressWarnings({
  trained = mvbc.trainer(
    Ybin ~ mercury + lead, ~ age, trainset$data,
    griddomain = seq(0, 2, length.out = 4), K = 3
  )
})
p_hat = mvbc.trainer.predict.p_hat(trained, testset$data, trained$minimas[["ssdr"]])
head(p_hat)
```

`mvbc.trainer.predict.Ybin_hat` *Predict outcomes*

Description

Predict $\hat{Y_{bin}}$ outcomes from training model

Usage

```
mvbc.trainer.predict.Ybin_hat(prior, p_hat)
```

Arguments

`prior`	estimated prior probability
`p_hat`	estimated $\hat{p}$ values

Value

integer vector of predicted binary outcomes (0/1)

mvbc.trainer.ssr *Sum of Squares residuals*

Description

Return sum-of-squares of Pearson and Deviance Residuals

Usage

```
mvbc.trainer.ssr(Ybin, p_hat, weights = NULL)
```

Arguments

Ybin	actual class values
p_hat	estimated $\hat{p}$ values
weights	optional vector of observation weights (e.g. survey weights); defaults to equal weights of 1 when NULL

Value

named numeric vector c(sspr, ssdr) with the sum-of-squares of the Pearson and Deviance residuals

plot.mvbc.sim.result *Plot instance of* mvbc.sim.result

Description

Plot instance of mvbc.sim.result

Usage

```
## S3 method for class 'mvbc.sim.result'
plot(x, ...)
```

Arguments

x	instance of the class mvbc.sim.model
...	pass thru to boxplot

Value

No return value, called for side effects

`plot.mvbc.tester.result`
Plot instance of `mvbc.tester.result`

Description

Plot instance of `mvbc.tester.result`

Usage

```
## S3 method for class 'mvbc.tester.result'
plot(x, ...)
```

Arguments

`x`	instance of the class `mvbc.tester.model`
`...`	pass thru to boxplot

Value

No return value, called for side effects

`summary.mvbc.sim.result`
Summarize instance of `mvbc.sim.result`

Description

Summarize instance of `mvbc.sim.result`

Usage

```
## S3 method for class 'mvbc.sim.result'
summary(object, ...)
```

Arguments

`object`	instance of the class `mvbc.sim.result`
`...`	pass thru nowhere

Value

No return value, called for side effects

summary.mvbc.simulator.model
Summarize instance of mvbc.simulator.model

Description

Summarize instance of mvbc.simulator.model

Usage

```
## S3 method for class 'mvbc.simulator.model'
summary(object, ...)
```

Arguments

object	instance of mvbc.simulator.model
...	pass thru to nowhere

Value

No return value, called for side effects

summary.mvbc.tester.result
Summarize instance of mvbc.tester.result

Description

Summarize instance of mvbc.tester.result

Usage

```
## S3 method for class 'mvbc.tester.result'
summary(object, ...)
```

Arguments

object	instance of the class mvbc.tester.model
...	pass thru nowhere

Value

No return value, called for side effects

summary.mvbc.trainer.model *Summarize instance of* mvbc.trainer.model

Description

Summarize instance of mvbc.trainer.model

Usage

```
## S3 method for class 'mvbc.trainer.model'
summary(object, ...)
```

Arguments

object	instance of the class mvbc.trainer.model
...	pass thru to nowhere

Value

No return value, called for side effects

Index