Package 'MIBRR'

June 6, 2018

Type Package

Title Multiple Imputation with Bayesian Regularized Regression

Version 0.0.0.9006

Date 2018-06-06

Author Kyle M. Lang [aut, crt]

Maintainer Kyle M. Lang <k.m.lang@uvt.nl>

Description This package implements a multiple imputation method that uses Bayesian regularized regression models as the elementary imputation methods.

License GPL-3 | file LICENSE

Depends mvtnorm, lavaan, mice, optimx, rlecuyer

Imports Rcpp (>= 0.11.2), RcppEigen (>= 0.3.2.2.0), methods

LinkingTo Rcpp, RcppEigen

URL http://github.com/kylelang/MIBRR

BugReports https://github.com/kylelang/MIBRR/issues

R topics documented:

MIBRR-package *Multiple Imputation with Bayesian Regularized Regression*

Description

This package implements a multiple imputation method that uses Bayesian regularized regression models as the elementary imputation methods.

Details

Index: This package was not yet installed at build time.

Author(s)

Kyle M. Lang [aut, crt]

Maintainer: Kyle M. Lang <k.m.lang@uvt.nl>

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

Li, Q. and Lin, N. (2010) The Bayesian Elastic Net. *Bayesian Analysis*, **5**(1), 151–170.

Park, T. and Casella, G. (2008) The Bayesian Lasso. *Journal of the American Statistical Association*, **103**, 681–686.

Zhao, Y., and Long, Q. (2013) Multiple imputation in the presence of high-dimensional data. *Statistical Methods in Medical Research*, **0**(0), 1–15.

Examples

```
data(mibrrExampleData)

mibenOut <- miben(data       = mibrrExampleData,
                  iterations = c(30, 10),
                  targetVars = c("y", paste0("x", c(1 : 3))),
                  ignoreVars = "idNum")

miblOut <- mibl(data       = mibrrExampleData,
                iterations = c(50, 10),
                targetVars = c("y", paste0("x", c(1 : 3))),
                ignoreVars = "idNum")

benOut <- ben(data       = mibrrExampleData,
              y          = "y",
              X          = setdiff(colnames(mibrrExampleData), c("y", "idNum")),
              iterations = c(30, 10)
              )
```

```
blOut <- bl(data       = mibrrExampleData,
            y          = "y",
            X          = setdiff(colnames(mibrrExampleData), c("y", "idNum")),
            iterations = c(50, 10)
            )
```

---

ben *Bayesian Elastic Net*

Description

This function will fit the Bayesian elastic net to (possibly) incomplete data.

Usage

```
ben(data,
    y,
    X            = NULL,
    iterations   = c(100, 10),
    sampleSizes  = list(rep(25, 2), rep(250, 2), rep(500, 2)),
    doMcem       = TRUE,
    lam1PriorPar = NULL,
    lam2PriorPar = NULL,
    missCode     = NA,
    verbose      = TRUE,
    seed         = NULL,
    userRng      = "",
    control      = list()
    )
```

Arguments

`data`	A, possibly incomplete, numeric data matrix or data frame to which the BEN is to be fit.
`y`	The column label for the outcome variable.
`X`	An optional character vector giving the column labels for the predictor variables. When `X = NULL` the target variable is regressed onto all other variables in `data`.
`iterations`	A two-element numeric vector giving the number of iterations to employ during the MCEM approximation and tuning phases, respectively. Defaults to `iterations = c(100, 10)`.
`sampleSizes`	A list or vector giving the desired Gibbs sample sizes (see Details for more information). Defaults to `sampleSizes = list(rep(25, 2), rep(250, 2), rep(500, 2))`.

doMcem	A logical switch: Should the model be estimated using Markov Chain Expectation Maximization (MCEM)? If doMcem = FALSE the model is estimated as a fully Bayesian model by assigning hyper-priors to the penalty parameters. Defaults to doMcem = TRUE.
lam1PriorPar	A two-element numeric vector giving the prior shape and rate parameters, respectively, for the squared LASSO penalty parameter's, λ_1^2, gamma prior.
lam2PriorPar	A two-element numeric vector giving the prior χ and ψ parameters, respectively, for the ridge penalty parameter's, λ_2, generalized inverse gaussian prior.
missCode	An optional integer-valued code used to flag the missing data in data. Should take a value that cannot naturally occur in data. Not needed when the missing data are coded as NA.
verbose	A logical switch: Should verbose output be printed to stdout? Defaults to verbose = TRUE.
seed	Either a positive, integer-valued seed a one-element character vector naming an extant R'Lecuyer RNG stream generated by the **rlecuyer** package (see "Details").
userRng	A character string giving the name of an **rlecuyer** RNG stream that is being actively used for random number generation in the current session. See "Details" for more information about how this argument is used.
control	A list of control parameters for the Gibbs sampler and penalty parameter optimization (see Details for more information).

Details

The sampleSizes argument must be a three-element list when doMcem = TRUE. In this case, the list must contain three two-element numeric (integer) vectors giving the number of MCMC draws to discard as burn-in and to retain, respectively, during the MCEM approximation, tuning, and sampling phases.

When doMcem = FALSE, only a single Gibbs sample is drawn, so thesampleSizes argument must be a two-element numeric (integer) vector giving the number of MCMC draws to discard as burn-in and the number to retain in the final Gibbs sample.

The **MIBRR** package uses **rlecuyer** to generate independent random number streams for all internal sub-processes. When the argument to seed is an integer, this value is used to seed the **rlecuyer** package via rlecuyer::.lec.SetPackageSeed(rep(seed, 6)). When the argument to seed names an extant **rlecuyer** stream, the internal RNG streams are seeded with the current state of the stream named in seed via rlecuyer::.lec.SetPackageSeed(rlecuyer::.lec.GetState(seed)). When seed = NULL, a transient seed is generated via seed <- round(runif(1) * 1e8). The value used to seed the **rlecuyer** package (along with its name, if any) is saved in the seed field of the MibrrFit object.

The **MIBRR** package uses **rlecuyer** random number streams internally. So, if an **rlecuyer** RNG stream, say "s0", is being used to generate random numbers in the current session, this stream's name should be given as the userRng argument. For example, if .lec.CurrentStream("s0") has been called and .lec.CurrentStreamEnd() has not been called, ben needs to be overtly notified of the non-standard RNG state via userRng = "s0". Doing so will tell ben to re-set "s0" as the active RNG stream on return. If no argument is provided for userRng, ben will attempt to leave the

RNG state as it was when ben was called, but no **rlecuyer** RNG streams will not be re-set for use in the current session.

control is a list containing the following named elements:

convThresh: The R-Hat value used to judge convergence. R-Hat values < convThresh arising from the final, retained Gibbs sample will trigger a warning.
Defaults to convThres = 1.1.

lambda1Starts: An optional numeric vector giving starting values for the LASSO penalty parameter, λ_1. Values are recycled to populate a vector with size = length(targetVars).
Defaults to rep(0.5, length(targetVars)).

lambda2Starts: An optional numeric vector giving starting values for the ridge penalty parameter, λ_2. Values are recycled to populate a vector with size = length(targetVars).
Defaults to rep(0.1 * nPreds, length(targetVars)), where nPreds is the number of predictors in the model.

usePcStarts: A logical switch: Use the starting values for λ_1 suggested by Park and Casella (2008)?
Defaults to usePcStarts = FALSE.

smoothingWindow: An integer giving the number of approximation phase Λ values to average over to get the starting $\Lambda's$ for the MCEM tuning phase. Setting smoothingWindow > 1 can facilitate convergence of the MCEM tuning phase when burn-in Λ estimates are very noisy. Ignored when doMcem = FALSE.
Defaults to smoothingWindow = min(10, ceiling(nApprox / 10)) where nApprox is the number of MCEM approximation iterations.

center: A logical switch: Should the data be centered before estimating the imputation model? When center = TRUE the data centers are added back to the imputed data before the function returns.
Defaults to center = TRUE.

scale: A logical switch: Should the predictor data be scaled to have unit variance before estimating the imputation model? When scale = TRUE imputed data are reverted to their original scaling before the function returns.
Defaults to scale = TRUE.

adaptScales: A logical switch: Should the target variables' scales be actively updated as part of imputation model estimation?
Defaults to adaptScales = TRUE.

minPredCor: The minimum correlation used by mice::quickpred when temporarily filling missing data before scaling or when filling missing data on covariates.
Defaults to minPredCor = 0.3.

miceIters: The number of iterations used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to miceIters = 10.

miceRidge: The ridge penalty used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to miceRidge = 1e-4.

miceMethod: The elementary imputation method used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to miceMethod = "pmm".

fimlStarts: A logical switch: Should the model moments from a saturated FIML model be used to scale the target variables? When `fimlStarts = TRUE`, the saturated model is estimated using **lavaan**.
Defaults to `fimlStarts = FALSE`.

optTraceLevel: A non-negative integer passed to the **optimx** `trace` argument. See **optimx** documentation for details.
Defaults to `optTraceLevel = 0`.

optCheckKkt: A logical flag: Should the Kuhn, Jarush, Tucker optimality conditions be checked when optimizing the penalty parameters?
Defaults to `optCheckKkt = TRUE`.

optMethod: A character vector giving the optimization method(s) used by **optimx** to estimate the penalty parameters. Possible options are "Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "nlm", "nlminb", "spg", "ucminf", "newuoa", "bobyqa", "nmkb", "hjkb", "Rcgmin", or "Rvmmin". When `length(optMethod) > 1`, **optimx**'s follow-on optimization is employed. See the **optimx** documentation for details.
Defaults to `optMethod = "L-BFGS-B"`.

optBoundLambda: A logical switch: Should the penalty parameters be bounded below by zero?
Defaults to `optBoundLambda = TRUE`.

checkConv: A logical switch: Should the stationary Gibbs samples be checked for convergence?
Defaults to `checkConv = TRUE`.

Value

A reference class object with class `MibrrFit`. This object contains a great deal of metadata and various pieces of output. Key output can be accessed via the convenience function `getParams`, and the `getField` function. See documentation for `MibrrFit` for more details.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Author(s)

Kyle M. Lang

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

Li, Q. and Lin, N. (2010) The Bayesian Elastic Net. *Bayesian Analysis*, **5**(1), 151–170.

See Also

`bl`, `optimx`, `getParams`, `getField`

Examples

```
data(mibrrExampleData)

## MCEM estimation:
benOut <- ben(data       = mibrrExampleData,
              y          = "y",
              X          = setdiff(colnames(mibrrExampleData), c("y", "idNum")),
              iterations = c(30, 10)
              )

## Fully Bayesian estimation:
benOut <- ben(data         = mibrrExampleData,
              y            = "y",
              X            = setdiff(colnames(mibrrExampleData), c("y", "idNum")),
              doMcem       = FALSE,
              sampleSizes  = c(500, 500),
              lam1PriorPar = c(1.0, 0.1),
              lam2PriorPar = c(1.0, 0.1)
              )
```

---

bl *Bayesian LASSO*

Description

This function will fit the Bayesian LASSO to (possibly) incomplete data.

Usage

```
bl(data,
   y,
   X            = NULL,
   iterations   = c(100, 10),
   sampleSizes  = list(rep(25, 2), rep(250, 2), rep(500, 2)),
   lam1PriorPar = NULL,
   doMcem       = TRUE,
   missCode     = NA,
   verbose      = TRUE,
   seed         = NULL,
   userRng      = "",
   control      = list()
   )
```

Arguments

data	A, possibly incomplete, numeric data matrix or data frame to which the BL is to be fit.
y	The column label for the outcome variable.

X	An optional character vector giving the column labels for the predictor variables. When X = NULL the target variable is regressed onto all other variables in data.
iterations	A two-element numeric vector giving the number of iterations to employ during the MCEM approximation and tuning phases, respectively. Defaults to iterations = c(100, 10).
sampleSizes	A list or vector giving the desired Gibbs sample sizes (see Details for more information). Defaults to sampleSizes = list(rep(25, 2), rep(250, 2), rep(500, 2)).
doMcem	A logical switch: Should the model be estimated using Markov Chain Expectation Maximization (MCEM)? If doMcem = FALSE the model is estimated as a fully Bayesian model by assigning hyper-priors to the penalty parameter. Defaults to doMcem = TRUE.
lam1PriorPar	A two-element numeric vector giving the prior shape and rate parameters, respectively, for the squared LASSO penalty parameter's, λ_1^2, gamma prior.
missCode	An optional integer-valued code used to flag the missing data in data. Should take a value that cannot naturally occur in data. Not needed when the missing data are coded as NA.
verbose	A logical switch: Should verbose output be printed to stdout? Defaults to verbose = TRUE.
seed	Either a positive, integer-valued seed a one-element character vector naming an extant R'Lecuyer RNG stream generated by the **rlecuyer** package (see "Details").
userRng	A character string giving the name of an **rlecuyer** RNG stream that is being actively used for random number generation in the current session. See "Details" for more information about how this argument is used.
control	A list of control parameters for the Gibbs sampler and penalty parameter optimization (see Details for more information).

Details

The sampleSizes argument must be a three-element list when doMcem = TRUE. In this case, the list must contain three two-element numeric (integer) vectors giving the number of MCMC draws to discard as burn-in and to retain, respectively, during the MCEM approximation, tuning, and sampling phases.

When doMcem = FALSE, only a single Gibbs sample is drawn, so thesampleSizes argument must be a two-element numeric (integer) vector giving the number of MCMC draws to discard as burn-in and the number to retain in the final Gibbs sample.

The **MIBRR** package uses **rlecuyer** to generate independent random number streams for all internal sub-processes. When the argument to seed is an integer, this value is used to seed the **rlecuyer** package via rlecuyer:::.lec.SetPackageSeed(rep(seed, 6)). When the argument to seed names an extant **rlecuyer** stream, the internal RNG streams are seeded with the current state of the stream named in seed via rlecuyer:::.lec.SetPackageSeed(rlecuyer:::.lec.GetState(seed)). When seed = NULL, a transient seed is generated via seed <- round(runif(1) * 1e8). The value used to seed the **rlecuyer** package (along with its name, if any) is saved in the seed field of the MibrrFit object.

The **MIBRR** package uses **rlecuyer** random number streams internally. So, if an **rlecuyer** RNG stream, say "s0", is being used to generate random numbers in the current session, this stream's name should be given as the `userRng` argument. For example, if `.lec.CurrentStream("s0")` has been called and `.lec.CurrentStreamEnd()` has not been called, `bl` needs to be overtly notified of the non-standard RNG state via `userRng = "s0"`. Doing so will tell `bl` to re-set "s0" as the active RNG stream on return. If no argument is provided for `userRng`, `bl` will attempt to leave the RNG state as it was when `bl` was called, but no **rlecuyer** RNG streams will not be re-set for use in the current session.

`control` is a list containing the following named elements:

convThresh: The R-Hat value used to judge convergence. R-Hat values < `convThresh` arising from the final, retained Gibbs sample will trigger a warning.
Defaults to `convThres = 1.1`.

lambda1Starts: An optional numeric vector giving starting values for the LASSO penalty parameter, λ_1. Values are recycled to populate a vector with `size = length(targetVars)`.
Defaults to `rep(0.5, length(targetVars))`.

usePcStarts: A logical switch: Use the starting values for λ_1 suggested by Park and Casella (2008)?
Defaults to `usePcStarts = FALSE`.

smoothingWindow: An integer giving the number of approximation phase Λ values to average over to get the starting $\Lambda's$ for the MCEM tuning phase. Setting `smoothingWindow` > 1 can facilitate convergence of the MCEM tuning phase when burn-in Λ estimates are very noisy.
Defaults to `smoothingWindow = min(10, ceiling(nApprox / 10))` where `nApprox` is the number of MCEM approximation iterations.

center: A logical switch: Should the data be centered before estimating the imputation model? When `center = TRUE` the data centers are added back to the imputed data before the function returns.
Defaults to `center = TRUE`.

scale: A logical switch: Should the predictor data be scaled to have unit variance before estimating the imputation model? When `scale = TRUE` imputed data are reverted to their original scaling before the function returns.
Defaults to `scale = TRUE`.

adaptScales: A logical switch: Should the target variables' scales be actively updated as part of imputation model estimation?
Defaults to `adaptScales = TRUE`.

minPredCor: The minimum correlation used by `mice::quickpred` when temporarily filling missing data before scaling or when filling missing data on covariates.
Defaults to `minPredCor = 0.3`.

miceIters: The number of iterations used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to `miceIters = 10`.

miceRidge: The ridge penalty used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to `miceRidge = 1e-4`.

miceMethod: The elementary imputation method used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to `miceMethod = "pmm"`.

fimlStarts: A logical switch: Should the model moments from a saturated FIML model be used to scale the target variables? When `fimlStarts = TRUE`, the saturated model is estimated using **lavaan**.
Defaults to `fimlStarts = FALSE`.

checkConv: A logical switch: Should the stationary Gibbs samples be checked for convergence?
Defaults to `checkConv = TRUE`.

Value

A reference class object with class `MibrrFit`. This object contains a great deal of metadata and various pieces of output. Key output can be accessed via the convenience function `getParams`, and the `getField` function. See documentation for `MibrrFit` for more details.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Author(s)

Kyle M. Lang

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

Park, T. and Casella, G. (2008) The Bayesian Lasso. *Journal of the American Statistical Association*, **103**, 681–686.

Zhao, Y., and Long, Q. (2013) Multiple imputation in the presence of high-dimensional data. *Statistical Methods in Medical Research*, **0**(0), 1–15.

See Also

`ben`, `getParams`, `getField`

Examples

```
data(mibrrExampleData)

## MCEM estimation:
blOut <- bl(data       = mibrrExampleData,
            y          = "y",
            X          = setdiff(colnames(mibrrExampleData), c("y", "idNum")),
            iterations = c(50, 10)
            )

## Fully Bayesian estimation:
```

```
blOut <- bl(data         = mibrrExampleData,
            y            = "y",
            X            = setdiff(colnames(mibrrExampleData), c("y", "idNum")),
            doMcem       = FALSE,
            sampleSizes  = c(500, 500),
            lam1PriorPar = c(1.0, 0.1)
            )
```

`getField` *Access arbitrary fields in a MibrrFit object.*

Description

This function will access any field in a fitted `MibrrFit` object using a familiar, 'inspect'-like interface.

Usage

```
getField(mibrrFit, what)
```

Arguments

`mibrrFit`	A fitted model object (with class `MibrrFit`) returned by `miben`, `mibl`.
`what`	The name of the field to extract (see Details for a list of the most interesting possibilities).

Details

This function can access any field in the `MibrrFit` object (see `MibrrFit` for a full list of possible fields), but the most interesting possibilities are the following:

data

rHats

lambdaHistory

gibbsOut

Value

The contents of whatever field is defined by the `what` argument.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Author(s)

Kyle M. Lang

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

See Also

miben, mibl, ben, bl

Examples

```
data(mibrrExampleData)

## Fit a Bayesian elastic net model:
benOut <- ben(data         = mibrrExampleData,
              y            = "y",
              X            = paste0("x", c(1 : 3)),
              sampleSizes  = c(500, 500),
              doMcem       = FALSE,
              lam1PriorPar = c(1.0, 0.1),
              lam2PriorPar = c(1.0, 0.1)
              )

## Extract the potential scale reduction factors:
benRHats <- getField(mibrrFit = benOut, what = "rHats")
```

---

getImpData *Generate multiply imputed datasets.*

Description

Given a fitted miben or mibl model, this funciton will replace missing values with imputations to generate a set of multiply imputed data.

Usage

```
getImpData(mibrrFit, nImps)
```

Arguments

mibrrFit	A fitted model object (with class MibrrFit) returned by miben, mibl.
nImps	The number of imputed datasets to return.

Value

A list containing `nImps` imputed datasets wherein the missing values in the target variables named in the `mibrrFit` object are replaced by random draws from the appropriate posterior predicted distributions.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Note

`nImps` cannot be larger than the posterior Gibbs sample size in the `mibrrFit` object.

Author(s)

Kyle M. Lang

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

See Also

`miben`, `mibl`

Examples

```
data(mibrrExampleData)

## Fit a Bayesian elastic net model:
mibenOut <- miben(data         = mibrrExampleData,
                  targetVars   = c("y", paste0("x", c(1 : 3))),
                  ignoreVars   = "idNum",
                  sampleSizes  = c(500, 500),
                  doMcem       = FALSE,
                  lam1PriorPar = c(1.0, 0.1),
                  lam2PriorPar = c(1.0, 0.1)
                  )

## Generate 25 imputed datasets:
mibenImps <- getImpData(mibrrFit = mibenOut, nImps = 25)
```

`getParams` *Extract posterior samples of model parameters.*

Description

Given a fitted `miben`, `mibl`, `ben`, or `bl` model, this funciton will extract the stationary Gibbs samples of the model parameters.

Usage

```
getParams(mibrrFit, target)
```

Arguments

`mibrrFit`	A fitted model object (with class `MibrrFit`) returned by `miben`, `mibl`.
`target`	The column label for the target variable whose parameter samples are to be extracted.

Value

A list containing the stationary Gibbs samples of the model parameters for the model wherein `target` was the outcome variable.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Note

If `doMcem = TRUE` the final optimized values of the penalty parameters, $\{\lambda_1, \lambda_2\}$, are returned. If `doMcem = FALSE`, the posterior samples of the penalty parameters are returned.

Author(s)

Kyle M. Lang

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

See Also

`miben`, `mibl`, `ben`, `bl`

Examples

```
data(mibrrExampleData)

## Fit a Bayesian elastic net model:
benOut <- ben(data         = mibrrExampleData,
              y            = "y",
              X            = paste0("x", c(1 : 3)),
              sampleSizes  = c(500, 500),
              doMcem       = FALSE,
              lam1PriorPar = c(1.0, 0.1),
              lam2PriorPar = c(1.0, 0.1)
              )

## Extract posterior parameter samples:
benPars <- getParams(mibrrFit = benOut, target = "y")
```

`miben` *Multiple Imputation with the Bayesian Elastic Net*

Description

This function implements MIBEN, a robust MICE-based multiple imputation scheme that employs the Bayesian elastic net as its elementary imputation method.

Usage

```
miben(data,
      targetVars   = NULL,
      ignoreVars   = NULL,
      iterations   = c(100, 10),
      sampleSizes  = list(rep(25, 2), rep(250, 2), rep(500, 2)),
      doMcem       = TRUE,
      lam1PriorPar = NULL,
      lam2PriorPar = NULL,
      missCode     = NA,
      verbose      = TRUE,
      seed         = NULL,
      userRng      = "",
      control      = list()
      )
```

Arguments

`data`	An incomplete, numeric data matrix or data frame for which to create the imputations.
`targetVars`	An optional character vector giving the column labels for the variables to be imputed. When `targetVars = NULL`, all variables not listed in `ignoreVars` are imputed.

`ignoreVars`	An optional character vector giving the column labels for those variables that should be excluded from the imputation model (e.g., ID variables).
`iterations`	A two-element numeric vector giving the number of iterations to employ during the MCEM approximation and tuning phases, respectively. Defaults to `iterations = c(100, 10)`.
`sampleSizes`	A list or vector giving the desired Gibbs sample sizes (see Details for more information). Defaults to `sampleSizes = list(rep(25, 2), rep(250, 2), rep(500, 2))`.
`doMcem`	A logical switch: Should the model be estimated using Markov Chain Expectation Maximization (MCEM)? If `doMcem = FALSE` the model is estimated as a fully Bayesian model by assigning hyper-priors to the penalty parameters. Defaults to `doMcem = TRUE`.
`lam1PriorPar`	A two-element numeric vector giving the prior shape and rate parameters, respectively, for the squared LASSO penalty parameter's, λ_1^2, gamma prior.
`lam2PriorPar`	A two-element numeric vector giving the prior χ and ψ parameters, respectively, for the ridge penalty parameter's, λ_2, generalized inverse Gaussian prior.
`missCode`	An optional integer-valued code used to flag the missing data in `data`. Should take a value that cannot naturally occur in `data`. Not needed when the missing data are coded as `NA`.
`verbose`	A logical switch: Should verbose output be printed to stdout? Defaults to `verbose = TRUE`.
`seed`	Either a positive, integer-valued seed a one-element character vector naming an extant R'Lecuyer RNG stream generated by the **rlecuyer** package (see "Details").
`userRng`	A character string giving the name of an **rlecuyer** RNG stream that is being actively used for random number generation in the current session. See "Details" for more information about how this argument is used.
`control`	A list of control parameters for the Gibbs sampler and penalty parameter optimization (see Details for more information).

Details

The `sampleSizes` argument must be a three-element list when `doMcem = TRUE`. In this case, the list must contain three two-element numeric (integer) vectors giving the number of MCMC draws to discard as burn-in and to retain, respectively, during the MCEM approximation, tuning, and sampling phases.

When `doMcem = FALSE`, only a single Gibbs sample is drawn, so the`sampleSizes` argument must be a two-element numeric (integer) vector giving the number of MCMC draws to discard as burn-in and the number to retain in the final Gibbs sample.

The **MIBRR** package uses **rlecuyer** to generate independent random number streams for all internal sub-processes. When the argument to `seed` is an integer, this value is used to seed the **rlecuyer** package via `rlecuyer::.lec.SetPackageSeed(rep(seed, 6))`. When the argument to `seed` names an extant **rlecuyer** stream, the internal RNG streams are seeded with the current state of the stream named in `seed` via `rlecuyer::.lec.SetPackageSeed(rlecuyer::.lec.GetState(seed))`. When `seed = NULL`, a transient seed is generated via `seed <- round(runif(1) * 1e8)`. The value

used to seed the **rlecuyer** package (along with its name, if any) is saved in the seed field of the MibrrFit object.

The **MIBRR** package uses **rlecuyer** random number streams internally. So, if an **rlecuyer** RNG stream, say "s0", is being used to generate random numbers in the current session, this stream's name should be given as the userRng argument. For example, if .lec.CurrentStream("s0") has been called and .lec.CurrentStreamEnd() has not been called, miben needs to be overtly notified of the non-standard RNG state via userRng = "s0". Doing so will tell miben to re-set "s0" as the active RNG stream on return. If no argument is provided for userRng, miben will attempt to leave the RNG state as it was when miben was called, but no **rlecuyer** RNG streams will not be re-set for use in the current session.

control is a list containing the following named elements:

convThresh: The R-Hat value used to judge convergence. R-Hat values < convThresh arising from the final, retained Gibbs sample will trigger a warning.
Defaults to convThres = 1.1.

lambda1Starts: An optional numeric vector giving starting values for the LASSO penalty parameter, λ_1. Values are recycled to populate a vector with size = length(targetVars).
Defaults to rep(0.5, length(targetVars)).

lambda2Starts: An optional numeric vector giving starting values for the ridge penalty parameter, λ_2. Values are recycled to populate a vector with size = length(targetVars).
Defaults to rep(0.1 * nPreds, length(targetVars)), where nPreds is the number of predictors in the imputation model.

usePcStarts: A logical switch: Use the starting values for λ_1 suggested by Park and Casella (2008)?
Defaults to usePcStarts = FALSE.

smoothingWindow: An integer giving the number of approximation phase Λ values to average over to get the starting $\Lambda's$ for the MCEM tuning phase. Setting smoothingWindow > 1 can facilitate convergence of the MCEM tuning phase when burn-in Λ estimates are very noisy.
Defaults to smoothingWindow = min(10, ceiling(nApprox / 10)) where nApprox is the number of MCEM approximation iterations.

center: A logical switch: Should the data be centered before estimating the imputation model? When center = TRUE the data centers are added back to the imputed data before the function returns.
Defaults to center = TRUE.

scale: A logical switch: Should the predictor data be scaled to have unit variance before estimating the imputation model? When scale = TRUE imputed data are reverted to their original scaling before the function returns.
Defaults to scale = TRUE.

adaptScales: A logical switch: Should the target variables' scales be actively updated as part of imputation model estimation?
Defaults to adaptScales = TRUE.

minPredCor: The minimum correlation used by mice::quickpred when temporarily filling missing data before scaling or when filling missing data on covariates.
Defaults to minPredCor = 0.3.

miceIters: The number of iterations used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to miceIters = 10.

miceRidge: The ridge penalty used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates. Defaults to `miceRidge = 1e-4`.

miceMethod: The elementary imputation method used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to `miceMethod = "pmm"`.

fimlStarts: A logical switch: Should the model moments from a saturated FIML model be used to scale the target variables? When `fimlStarts = TRUE`, the saturated model is estimated using **lavaan**.
Defaults to `fimlStarts = FALSE`.

preserveStructure: A logical switch: Should the data columns be returned in the same order as submitted?
Defaults to `preserveStructure = TRUE`.

optTraceLevel: A non-negative integer passed to the **optimx** `trace` argument. See **optimx** documentation for details.
Defaults to `optTraceLevel = 0`.

optCheckKkt: A logical flag: Should the Kuhn, Jarush, Tucker optimality conditions be checked when optimizing the penalty parameters?
Defaults to `optCheckKkt = TRUE`.

optMethod: A character vector giving the optimization method(s) used by **optimx** to estimate the penalty parameters. Possible options are "Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "nlm", "nlminb", "spg", "ucminf", "newuoa", "bobyqa", "nmkb", "hjkb", "Rcgmin", or "Rvmmin". When `length(optMethod) > 1`, **optimx**'s follow-on optimization is employed. See the **optimx** documentation for details.
Defaults to `optMethod = "L-BFGS-B"`.

optBoundLambda: A logical switch: Should the penalty parameters be bounded below by zero?
Defaults to `optBoundLambda = TRUE`.

checkConv: A logical switch: Should the stationary Gibbs samples be checked for convergence?
Defaults to `checkConv = TRUE`.

Value

A reference class object with class `MibrrFit`. This object contains a great deal of metadata and various pieces of output. Key output can be accessed via the convenience functions `getImpData` and `getParams` as well as the `getField` function. See documentation for `MibrrFit` for more details.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Author(s)

Kyle M. Lang

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

Li, Q. and Lin, N. (2010) The Bayesian Elastic Net. *Bayesian Analysis*, **5**(1), 151–170.

See Also

mibl, optimx, getImpData, getParams, getField

Examples

```
data(mibrrExampleData)

## MCEM estimation:
mibenOut <- miben(data       = mibrrExampleData,
                  iterations = c(30, 10),
                  targetVars = c("y", paste0("x", c(1 : 3))),
                  ignoreVars = "idNum")

## Fully Bayesian estimation:
mibenOut <- miben(data         = mibrrExampleData,
                  targetVars   = c("y", paste0("x", c(1 : 3))),
                  ignoreVars   = "idNum",
                  sampleSizes  = c(500, 500),
                  doMcem       = FALSE,
                  lam1PriorPar = c(1.0, 0.1),
                  lam2PriorPar = c(1.0, 0.1)
                  )
```

---

mibl *Multiple Imputation with the Bayesian LASSO*

Description

This function will implement a MICE-based multiple imputation scheme that employs the Bayesian LASSO as its elementary imputation method.

Usage

```
mibl(data,
     targetVars   = NULL,
     ignoreVars   = NULL,
     iterations   = c(100, 10),
     sampleSizes  = list(rep(25, 2), rep(250, 2), rep(500, 2)),
     doMcem       = TRUE,
     lam1PriorPar = NULL,
     missCode     = NA,
     verbose      = TRUE,
```

```
    seed        = NULL,
    userRng     = "",
    control     = list()
    )
```

Arguments

data	An incomplete, numeric data matrix or data frame for which to create the imputations.
targetVars	An optional character vector giving the column labels for the variables to be imputed. When `targetVars = NULL`, all variables not listed in `ignoreVars` are imputed.
ignoreVars	An optional character vector giving the column labels for those variables that should be excluded from the imputation model (e.g., ID variables).
iterations	A two-element numeric vector giving the number of iterations to employ during the MCEM approximation and tuning phases, respectively. Defaults to `iterations = c(100, 10)`.
sampleSizes	A list or vector giving the desired Gibbs sample sizes (see Details for more information). Defaults to `sampleSizes = list(rep(25, 2), rep(250, 2), rep(500, 2))`.
doMcem	A logical switch: Should the model be estimated using Markov Chain Expectation Maximization (MCEM)? If `doMcem = FALSE` the model is estimated as a fully Bayesian model by assigning hyper-priors to the penalty parameters. Defaults to `doMcem = TRUE`.
lam1PriorPar	A two-element numeric vector giving the prior shape and rate parameters, respectively, for the squared LASSO penalty parameter's, λ_1^2, gamma prior.
missCode	An optional integer-valued code used to flag the missing data in `data`. Should take a value that cannot naturally occur in `data`. Not needed when the missing data are coded as `NA`.
verbose	A logical switch: Should verbose output be printed to stdout? Defaults to `verbose = TRUE`.
seed	Either a positive, integer-valued seed a one-element character vector naming an extant R'Lecuyer RNG stream generated by the **rlecuyer** package (see "Details").
userRng	A character string giving the name of an **rlecuyer** RNG stream that is being actively used for random number generation in the current session. See "Details" for more information about how this argument is used.
control	A list of control parameters for the Gibbs sampler and penalty parameter optimization (see Details for more information).

Details

The `sampleSizes` argument must be a three-element list when `doMcem = TRUE`. In this case, the list must contain three two-element numeric (integer) vectors giving the number of MCMC draws to discard as burn-in and to retain, respectively, during the MCEM approximation, tuning, and sampling phases.

When doMcem = FALSE, only a single Gibbs sample is drawn, so thesampleSizes argument must be a two-element numeric (integer) vector giving the number of MCMC draws to discard as burn-in and the number to retain in the final Gibbs sample.

The **MIBRR** package uses **rlecuyer** to generate independent random number streams for all internal sub-processes. When the argument to seed is an integer, this value is used to seed the **rlecuyer** package via rlecuyer::.lec.SetPackageSeed(rep(seed, 6)). When the argument to seed names an extant **rlecuyer** stream, the internal RNG streams are seeded with the current state of the stream named in seed via rlecuyer::.lec.SetPackageSeed(rlecuyer::.lec.GetState(seed)). When seed = NULL, a transient seed is generated via seed <- round(runif(1) * 1e8). The value used to seed the **rlecuyer** package (along with its name, if any) is saved in the seed field of the MibrrFit object.

The **MIBRR** package uses **rlecuyer** random number streams internally. So, if an **rlecuyer** RNG stream, say "s0", is being used to generate random numbers in the current session, this stream's name should be given as the userRng argument. For example, if .lec.CurrentStream("s0") has been called and .lec.CurrentStreamEnd() has not been called, mibl needs to be overtly notified of the non-standard RNG state via userRng = "s0". Doing so will tell mibl to re-set "s0" as the active RNG stream on return. If no argument is provided for userRng, mibl will attempt to leave the RNG state as it was when mibl was called, but no **rlecuyer** RNG streams will not be re-set for use in the current session.

control is a list containing the following named elements:

convThresh: The R-Hat value used to judge convergence. R-Hat values < convThresh arising from the final, retained Gibbs sample will trigger a warning.
Defaults to convThres = 1.1.

lambda1Starts: An optional numeric vector giving starting values for the LASSO penalty parameter, λ_1. Values are recycled to populate a vector with size = length(targetVars).
Defaults to rep(0.5, length(targetVars)).

usePcStarts: A logical switch: Use the starting values for λ_1 suggested by Park and Casella (2008)?
Defaults to usePcStarts = FALSE.

smoothingWindow: An integer giving the number of approximation phase Λ values to average over to get the starting $\Lambda's$ for the MCEM tuning phase. Setting smoothingWindow > 1 can facilitate convergence of the MCEM tuning phase when burn-in Λ estimates are very noisy.
Defaults to smoothingWindow = min(10, ceiling(nApprox / 10)) where nApprox is the number of MCEM approximation iterations.

center: A logical switch: Should the data be centered before estimating the imputation model? When center = TRUE the data centers are added back to the imputed data before the function returns.
Defaults to center = TRUE.

scale: A logical switch: Should the predictor data be scaled to have unit variance before estimating the imputation model? When scale = TRUE imputed data are reverted to their original scaling before the function returns.
Defaults to scale = TRUE.

adaptScales: A logical switch: Should the target variables' scales be actively updated as part of imputation model estimation?
Defaults to adaptScales = TRUE.

minPredCor: The minimum correlation used by `mice::quickpred` when temporarily filling missing data before scaling or when filling missing data on covariates.
Defaults to `minPredCor = 0.3`.

miceIters: The number of iterations used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to `miceIters = 10`.

miceRidge: The ridge penalty used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to `miceRidge = 1e-4`.

miceMethod: The elementary imputation method used by the **mice** package when temporarily filling missing data before scaling or filling missing data on covariates.
Defaults to `miceMethod = "pmm"`.

fimlStarts: A logical switch: Should the model moments from a saturated FIML model be used to scale the target variables? When `fimlStarts = TRUE`, the saturated model is estimated using **lavaan**.
Defaults to `fimlStarts = FALSE`.

preserveStructure: A logical switch: Should the data columns be returned in the same order as submitted?
Defaults to `preserveStructure = TRUE`.

checkConv: A logical switch: Should the stationary Gibbs samples be checked for convergence?
Defaults to `checkConv = TRUE`.

Value

A reference class object with class `MibrrFit`. This object contains a great deal of metadata and various pieces of output. Key output can be accessed via the convenience functions `getImpData` and `getParams` as well as the `getField` function. See documentation for `MibrrFit` for more details.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Author(s)

Kyle M. Lang

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

Park, T. and Casella, G. (2008) The Bayesian Lasso. *Journal of the American Statistical Association*, **103**, 681–686.

Zhao, Y., and Long, Q. (2013) Multiple imputation in the presence of high-dimensional data. *Statistical Methods in Medical Research*, **0**(0), 1–15.

See Also

miben, getImpData, getParams, getField

Examples

```
data(mibrrExampleData)

## MCEM estimation:
miblOut <- mibl(data       = mibrrExampleData,
                iterations = c(50, 10),
                targetVars = c("y", paste0("x", c(1 : 3))),
                ignoreVars = "idNum")

## Fully Bayesian estimation:
miblOut <- mibl(data         = mibrrExampleData,
                targetVars   = c("y", paste0("x", c(1 : 3))),
                ignoreVars   = "idNum",
                sampleSizes  = c(500, 500),
                doMcem       = FALSE,
                lam1PriorPar = c(1.0, 0.1)
                )
```

---

mibrrExampleData *Example Dataset for the MIBRR package.*

Description

Toy data generated as in Experiment 1 of Lang (2015).

Usage

```
data("mibrrExampleData")
```

Format

A data frame with 200 observations on the following 17 variables.

idNum: "Participant" ID Number.

y: Outcome Variable. Contains 20% MAR missingness.

x1–x3: Substantive predictors. Contain 20% MAR missingness.

z1–z12: Exogenous auxiliary variables. Contain 10% MCAR missingness.

Details

These data are only simulated toy data; they have no true meaning. This is one of the datasets generated as part of the Monte Carlo simulation study used to conduct Experiment 1 of my dissertation. The missingness on {y, X} is caused by a linear combination of two randomly selected elements of {Z}, and half of the elements in {Z} have no association with {y, X} (see Lang, 2015, for more details).

Source

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

`MibrrFit-class` *Class* `"MibrrFit"`

Description

The `MibrrFit` class is a reference class object that acts as the general return object for the `MIBRR` package.

Extends

All reference classes extend and inherit methods from `"envRefClass"`.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Methods

The `MibrrFit` class contains many private methods that are not meant to be directly accessed by end users. Adventurous users can refer to the source code of the class' definition for details.

Any field in the `MibrrFit` object can be accessed via the `getField` function. The most interesting fields are described in the `getField` function's documentation.

After running `miben` or `mibl`, a set of imputed datasets can be generated with the `getImpData` function.

After running any of the primary modeling functions (i.e., `miben`, `mibl`, `ben`, `bl`), the model parameters' posterior Gibbs samples can be extracted via the `getParams` function and predictions can be generated via the `postPredict` function.

Fields

The `MibrrFit` class contains the following fields (which are not intended to be directly access by users). Interesting data, results, and metadata can be extracted from a fitted `MibrrFit` object via the functions described in the preceding "methods" section.

`data`**:** Object of class `data.frame` ~~

`targetVars`**:** Object of class `character` ~~

`ignoreVars`**:** Object of class `character` ~~

`iterations`**:** Object of class `integer` ~~

`sampleSizes`: Object of class `list` ~~

`missCode`: Object of class `integer` ~~

`seed`: Object of class `list` ~~

`doImp`: Object of class `logical` ~~

`doMcem`: Object of class `logical` ~~

`doBl`: Object of class `logical` ~~

`checkConv`: Object of class `logical` ~~

`verbose`: Object of class `logical` ~~

`convThresh`: Object of class `numeric` ~~

`lambda1Starts`: Object of class `numeric` ~~

`lambda2Starts`: Object of class `numeric` ~~

`l1Pars`: Object of class `numeric` ~~

`l2Pars`: Object of class `numeric` ~~

`usePcStarts`: Object of class `logical` ~~

`smoothingWindow`: Object of class `integer` ~~

`center`: Object of class `logical` ~~

`scale`: Object of class `logical` ~~

`adaptScales`: Object of class `logical` ~~

`minPredCor`: Object of class `numeric` ~~

`miceIters`: Object of class `integer` ~~

`miceRidge`: Object of class `numeric` ~~

`miceMethod`: Object of class `character` ~~

`fimlStarts`: Object of class `logical` ~~

`preserveStructure`: Object of class `logical` ~~

`optTraceLevel`: Object of class `integer` ~~

`optCheckKkt`: Object of class `logical` ~~

`optMethod`: Object of class `character` ~~

`optBoundLambda`: Object of class `logical` ~~

`dataMeans`: Object of class `numeric` ~~

`dataScales`: Object of class `numeric` ~~

`gibbsOut`: Object of class `list` ~~

`ignoredColumns`: Object of class `data.frame` ~~

`rawNames`: Object of class `character` ~~

`impRowsPool`: Object of class `integer` ~~

`missList`: Object of class `list` ~~

`nChains`: Object of class `integer` ~~

`rHats`: Object of class `list` ~~

`lambdaMat`: Object of class `matrix` ~~

`lambdaHistory`: Object of class `list` ~~

`betaStarts`: Object of class `matrix` ~~

`tauStarts`: Object of class `matrix` ~~

`sigmaStarts`: Object of class `numeric` ~~

`userMissCode`: Object of class `logical` ~~

`missCounts`: Object of class `integer` ~~

`nTargets`: Object of class `integer` ~~

`nVar`: Object of class `integer` ~~

`nPreds`: Object of class `integer` ~~

`nObs`: Object of class `integer` ~~

`totalIters`: Object of class `integer` ~~

Note

`MibrrFit` is a mutable *reference class* object, so functions in the `MIBRR` package that take a `MibrrFit` object as input will, generally, alter the state of that input object.

If you need to preserve a snapshot of a `MibrrFit` object, use the `copy` member function (e.g., `snapshot <- mibrrFit$copy()`).

Author(s)

Kyle M. Lang

See Also

`getImpData`, `getParams`, `getField`, `postPredict`

Examples

```
showClass("MibrrFit")
```

`mibrrL` *Print License for* **mibrr**

Description

Print the license under which **mibrr** is distibuted (i.e., the GPL-3).

Usage

```
mibrrL()
```

Value

Print the GPL-3 to stdout.

Author(s)

Kyle M. Lang

Examples

```
mibrrL()
```

---

mibrrW *Print Warranty Statement for* **mibrr**

Description

Print the sections of the GPL-3 that describe the warranty (or complete lack thereof) for **mibrr**.

Usage

```
mibrrW()
```

Value

Text giving the warranty-specific sections of the GPL-3.

Author(s)

Kyle M. Lang

Examples

```
mibrrW()
```

postPredict *Generate Posterior Predictions from MIBRR Models*

Description

This function will generate posterior predictive draws form models fit using `miben`, `mibl`, `ben`, or `bl`.

Usage

```
postPredict(mibrrFit, newData, targetVars = NULL, nDraws = 0)
```

Arguments

`mibrrFit`	A fitted model object (with class `MibrrFit`) returned by `miben`, `mibl`, `ben`, or `bl`.
`newData`	A data.frame containing new predictor data from which to generate the posterior predictions.
`targetVars`	An optional character vector giving the column labels for the outcome variables for which to generate posterior predictions. When `targetVar = NULL` predictions are generated for all target variables contained in `mibrrFit`.
`nDraws`	The number of posterior predictive draws to return. If `nDraws = N > 0` the draws are returned in an N-column matrix where each column contains the posterior predictions generated by a random sample of model parameters. If `nDraws = 0` predictions are generated using the posterior modes (i.e., MAP scores) of the model paramters. If `nDraws < 0` predictions are generated using the posterior means (i.e., EAP scores) of the model parameters. Defaults to `nDraws = 0`.

Value

A list containing the posterior predictive draws for each target variable defined in `targetVar` or for all target variables in `mibrrFit` when `targetVar = NULL`.

Warning

This function is in a highly unstable *alpha* level of development. Please anticipate frequent—and dramatic—changes to the functionality and user interface.

You have been granted access to this package for evaluation purposes, only. This function is **absolutely not** ready for use in real-world analyses!

Note

The column names of `newData` must contain the column names of all variables used to estimate the model represented by `mibrrFit`.

Author(s)

Kyle M. Lang

References

Lang, K. M. (2015) *MIBEN: Multiple imputation with the Bayesian elastic net* (Unpublished doctoral dissertation). University of Kansas.

See Also

miben, mibl, ben, bl

Examples

```
data(predictData)

## Fit a Bayesian elastic net model:
benOut <- ben(data       = predictData$train,
              y          = "agree",
              X          = setdiff(colnames(predictData$train), "agree"),
              iterations = c(30, 10)
              )

## Generate 10 posterior predictions for 'y':
benPred <- postPredict(mibrrFit = benOut,
                       newData  = predictData$test,
                       nDraws   = 10)

## Generate posterior predictions for 'y' using MAP scores (the default):
benPred <- postPredict(mibrrFit = benOut,
                       newData  = predictData$test,
                       nDraws   = 0)

## Generate posterior predictions for 'y' using EAP scores:
benPred <- postPredict(mibrrFit = benOut,
                       newData  = predictData$test,
                       nDraws   = -1)

## Fit chained Bayesian elastic net models to support MI:
mibenOut <- miben(data = predictData$incomplete, iterations = c(30, 10))

## Generate posterior predictions from elementary imputation models:
mibenPred <- postPredict(mibrrFit = mibenOut, newData = predictData$test)
```

---

predictData *Example Dataset to Demonstrate Prediction with the MIBRR Package.*

Description

These data can be used to demonstrate the prediction capabilities of ben, bl, miben, and mibl.

Usage

```
data("predictData")
```

Format

A three-element list containing the data frames described below.

All three data frames contain the same 11 variables. Scores on five personality dimensions (i.e., agreeableness, conscientiousness, extroversion, neuroticism, and openness to experience). Participants' ages, genders, and a set of four dummy codes indicating their level of educational achievement.

incomplete: An incomplete dataset with 500 observations of 11 variables. The five personality scores in this dataset are each subject of approximately 20% missing data.

test: A testing dataset with 50 observations of 11 variables.

train: A training dataset with 500 observations of 11 variables.

Details

These data are derived from the `bfi` data provided by the **psych** package. I scored the five personality constructs, dummy coded the covariates, and sub-sampled 500 observations. The incomplete dataset had 20% MAR missing data imposed on the personality constructs by defining the reponse propensity as a function of the fully observed covariates.

Source

Revelle, W. (2017) psych: Procedures for Personality and Psychological Research, Northwestern University, Evanston, Illinois, USA, https://CRAN.R-project.org/package=psych Version = 1.7.8.

Index