Package 'SURF'

October 15, 2018

Type Package

Title Some Useful R Functions

Version 0.0.0.9004

Date 2018-10-15

Author Kyle M. Lang

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

Description These are several useful functions that I find myself using often (read: frequently re-implementing), so I will package them for easy access/dissemination.

Depends mvtnorm

R topics documented:

`SURF-package` *Some Useful R Functions*

Description

These are several useful functions that I find myself using often (read: frequently re-implementing), so I will package them for easy access/dissemination.

Details

The DESCRIPTION file: This package was not yet installed at build time.

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

Author(s)

Kyle M. Lang

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

Examples

```
## Simulate regression data:
testData <- simRegData(nObs  = 100,
                       nVars = 10,
                       r2    = 0.5,
                       sigma = 0.2,
                       beta  = matrix(c(0.25, rep(0.75, 10)))
                       )

## Impose missing data
missData <- imposeMissData(data     = testData,
                           targets  = list(mar  = c("y", "x1"),
                                           mcar = c("x2", "x3"),
                                           mnar = c("x4", "x5")
                                           ),
                            preds   = c("x8", "x9", "x10"),
                            pm      = list(mar = 0.2, mcar = 0.1, mnar = 0.1),
                            snr     = list(mar = 5, mnar = 2.5),
                           pattern = "random")

## Plot imputed vs. observed values:
data(testImps)

plotImps(impList = testImps$impList,
         rMat    = testImps$rMat,
         typeVec = testImps$typeVec)
```

`calcMode` *Find the Modal Value of a Vector*

Description

This function will find the most commonly occurring value (i.e., the mode) in a vector.

Usage

```
calcMode(x, discrete = TRUE)
```

Arguments

`x`	A vector for which to find the mode.
`discrete`	A logical switch indicate whether `x` is a categorical (i.e., `discrete = TRUE`) or continuous (i.e., `discrete = FALSE`) variable.

Details

In the event of multiple modes, ties are broken randomly. When `x` is numeric, the return value will be a length one numeric vector, otherwise, the return value will be a length one character vector.

When `discrete = FALSE` the modal value is estimated via kernel density estimation by applying `stats::density` to `x`.

Value

The modal value in `x`.

Author(s)

Kyle M. Lang

Examples

```
## Find mode of an integer vector:
x1 <- sample(c(1 : 5), 100, TRUE)
calcMode(x1)

## Find mode of a factor:
x2 <- factor(sample(c(1 : 5), 100, TRUE))
calcMode(x2)

## Find mode of a character vector:
x3 <- sample(letters[1 : 5], 100, TRUE)
calcMode(x3)

## Find the mode of a continuous variable:
x4 <- rnorm(1000, 5, 2.5)
calcMode(x4, discrete = FALSE)
```

`f2n` *Safely Cast a Factor to a Numeric Vector*

Description

This function will type-cast a factor to a numeric vector without mangling the levels when those levels are zero-indexed integers.

Usage

```
f2n(x)
```

Arguments

`x`	A factor to be converted.

Details

R indexes factor levels from 1, so type-casting factors that have zero-indexed integer labels using `as.numeric` can produce unanticipated results. This function will cast such factors to zero-indexed numeric vectors wherein the original factor *labels* are mapped to numeric values (rather than the inherent factor *levels* being so mapped).

Value

A numeric vector containing the type-cast version of `x`.

Author(s)

Kyle M. Lang

Examples

```
## Generate a factor:
x1 <- factor(c(0 : 5))
x1

## Naive type-casting:
as.numeric(x1)

## Safe type-casting:
f2n(x1)
```

`imposeMissData` *Impose Missing Data*

Description

Impose missing data according to MAR, MCAR, and MNAR mechanisms.

Usage

```
imposeMissData(data,
               targets,
               preds,
               pm,
               snr,
               pattern = "random")
```

Arguments

`data`	A data.frame wherein missing data are to be imposed.
`targets`	A named list with slots "mar", "mcar", and "mnar" containing character vectors giving the column labels for variables onto which missing at random, missing completely at random, and missing not at random data, respectively, shall be imposed.
`preds`	A character vector giving the column labels for predictors of the MAR missingness.
`pm`	A named list with slots "mar", "mcar", and "mnar" containing real numbers in [0, 1) giving the proportions of missing at random, missing completely at random, and missing not at random data, respectively, to generate.
`snr`	A named list with slots "mar" and "mnar" containing real numbers giving the signal-to-noise ratios of the probit regression models used to impose missing at random and missing not at random data, respectively.
`pattern`	A character vector indicating in what parts of the missing data predictors' distributions MAR and MNAR missing data should be imposed. Legal keywords are: "low" = impose missing in the negative tail of the predictor, "high" = impose missing in the positive tail of the predictor, "center" = impose missing in the center of the predictor, "tails" = impose missing in both tails of the predictor. The `pattern` arugment can also be the special keyword "random" which will cause the function to randomly sample from the four preceding possiblities for each target variable.

Details

MCAR missing data is imposed by generating a random Bernoulli flag variable for each target variable with probablity of success equal to `pm$mcar`.

MAR missing data is imposed via a noisy probit regression model wherein the weighted sum of the columns listed in `preds` are used to predict the response propensity.

MNAR missing data is imposed via the same procedure as MAR missing data but the missing data predictor is taken to be the target variable itself.

Value

A two-element list with the following slots:

data: The incomplete version of `data`

pattern: A character vector showing which `pattern` was used for each target varaible.

Note

Due to the stochastic nature of the missing data simulation, the actual proportions of missing data will only equal the values provided for `pm` asymptotically.

Author(s)

Kyle M. Lang

See Also

`simRegData`

Examples

```
## Simulate some data:
testData <- simRegData(nObs  = 100,
                       nVars = 10,
                       r2    = 0.5,
                       sigma = 0.2,
                       beta  = matrix(c(0.25, rep(0.75, 10)))
                       )

## Impose missing data:
missData <- imposeMissData(data     = testData,
                           targets  = list(mar  = c("y", "x1"),
                                           mcar = c("x2", "x3"),
                                           mnar = c("x4", "x5")
                                           ),
                            preds   = c("x8", "x9", "x10"),
                            pm      = list(mar = 0.2, mcar = 0.1, mnar = 0.1),
                            snr     = list(mar = 5, mnar = 2.5),
                            pattern = "random")
```

`plotImps` *Plot Imputed vs. Observed Values*

Description

This function will generate plots of imputed versus observed values in multiply imputed data. These plots can be examined to "sanity-check" the imputation procedure.

Usage

```
plotImps(impList,
         rMat,
         typeVec,
         targetVar   = NULL,
         interactive = FALSE)
```

Arguments

`impList`	A list of multiply imputed datasets.
`rMat`	A logical pattern matrix flagging missing values in the orginal data used to generate the imputed datasets in `impList`. Note that `TRUE` flags missing cells and `FALSE` flags observed cells.

`typeVec`	A character vector with `length(typeVec) = ncol(impList[[1]])` giving the measurement levels of the variables in `impList`. Two values are recognized: "cat" = a categorical variable (i.e., nominal or ordinal) and "con" = a continuous variable (i.e., interval or ratio).
`targetVar`	An optional character vector giving the column names for variables to plot. When `targetVar = NULL` all variables with imputed values are plotted.
`interactive`	A logical flag: Should the cycle through all plotted variables by prompting the user to continue after generating each plot?

Value

Used for its side-effects.

Author(s)

Kyle M. Lang

Examples

```
data(testImps)

plotImps(impList = testImps$impList,
         rMat    = testImps$rMat,
         typeVec = testImps$typeVec)
```

`rangeNorm` *Range Normalize a Vector*

Description

This function will standardize a vector so that all transformed values exist between user-defined minimum and maximum values.

Usage

```
rangeNorm(x, oldMin = min(x), oldMax = max(x), newMin = 0.0, newMax = 1.0)
```

Arguments

`x`	A numeric vector to be standardized.
`oldMin`	An optional real number giving the minimum possible value of `x`.
`oldMax`	An optional real number giving the maximum possible value of `x`.
`newMin`	An optional real number giving the minimum possible value of the transformed input vector.
`newMax`	An optional real number giving the maximum possible value of the transformed input vector.

Value

A numeric vector containing the range-normalized version of `x`.

Author(s)

Kyle M. Lang

Examples

```
x0 <- runif(100, -3, 5)
x1 <- rangeNorm(x = x0)

range(x1)
```

---

simCovData *Simulate Data with Known Covariance Structure*

Description

This function will simulation simple data with a known covariance structure.

Usage

```
simCovData(nObs,
           sigma,
           nVars  = ncol(sigma),
           means  = 0.0,
           scales = 1.0)
```

Arguments

`nObs`	An integer giving the number of rows to simulate.
`sigma`	Either a numeric matrix giving the covariance matrix of the predictor variables or a length-one numeric vector with value in [-1.0, 1.0]. In the latter case, `sigma` gives the correlation between the predictors (i.e., the degree of collinearity).
`nVars`	An integer giving the number of (possibly latent) predictor variables to simulate. Defaults to `ncol(sigma)`.
`means`	A numeric vector of predictor means. Recycled when `length(means) = 1` to match `nVars`, otherwise `length(means)` must equal `nVars`.
`scales`	A numeric vector of predictor scales. Recycled when `length(scales) = 1` to match `nVars`, otherwise `length(scales)` must equal `nVars`. Ignored when `sigma` is a matrix.

Details

If `sigma` is a length-one vector, a covariance matrix is constructed from the values provided for `sigma` and `scales`. In this case, `sigma` gives the *correlation* between predictors (not the covariance) and `scales` gives the predictors' standard deviations.

Value

An `nObs` by `nVars` data.frame of simulated data.

Note

The column labels of the simulated data will be `paste0("x", 1 : nVars)`.

Author(s)

Kyle M. Lang

See Also

`imposeMissData` `simRegData`

Examples

```
## Specify 'sigma' as a full covariance matrix:
sigma       <- matrix(0.3, 10, 10)
diag(sigma) <- 1.0

testData <- simCovData(nObs = 100, sigma = sigma)

## Specify 'sigma' as an inter-predictor correlation:
testData <- simCovData(nObs = 100, nVars = 10, sigma = 0.2)
```

`simRegData` *Simulate Regression Data*

Description

This function will simulation regression data with known R-Squared, inter-predictor correlation, and latent grouping structure among the predictors.

Usage

```
simRegData(nObs,
           r2,
           sigma,
           beta,
           nVars          = ncol(sigma),
           means          = 0.0,
           scales          = 1.0,
           itemsPerPred    = 1,
           predReliability = 0.8)
```

Arguments

`nObs`	An integer giving the number of rows to simulate.
`r2`	A real number in [0, 1]. The R-Squared of the data generating model. That is, what proportion of variability in the outcome should be explained by the predictors.
`sigma`	Either a numeric matrix giving the covariance matrix of the predictor variables or a length-one numeric vector with value in [-1.0, 1.0]. In the latter case, `sigma` gives the correlation between the predictors (i.e., the degree of collinearity).
`beta`	An numeric matrix of regression coefficients with `dim(beta) = c(nVars, 1)`. Note that the first element is taken to be the intercept.
`nVars`	An integer giving the number of (possibly latent) predictor variables to simulate. Defaults to `ncol(sigma)`.
`means`	A numeric vector of predictor means. Recycled when `length(means) = 1` to match `nVars`, otherwise `length(means)` must equal `nVars`.
`scales`	A numeric vector of predictor scales. Recycled when `length(scales) = 1` to match `nVars`, otherwise `length(scales)` must equal `nVars`. Ignored when `sigma` is a matrix.
`itemsPerPred`	An integer giving the number of observed items used to define each latent predictor. When `itemsPerPred = 1`, no latent structure is imposed on the predictors.
`predReliability`	A real number in [0, 1]. When `itemsPerPred > 1`, `predReliability` defines the proportion of reliable variance among the indicators of each latent predictor. That is, the proportion of shared variance among each latent predictor's observed indicators.

Details

If `sigma` is a length-one vector, a covariance matrix is constructed from the values provided for `sigma` and `scales`. In this case, `sigma` gives the *correlation* between predictors (not the covariance) and `scales` gives the predictors' standard deviations.

Value

An `nObs` by `nVars * itemsPerPred + 1` data.frame of simulated data.

Note

The column labels of the simulated data will be `c("y", paste0("x", 1 : nVars * itemsPerPred))`.

Author(s)

Kyle M. Lang

See Also

imposeMissData simCovData

Examples

```
## Specify 'sigma' as a full covariance matrix:
sigma       <- matrix(0.3, 10, 10)
diag(sigma) <- 1.0

testData <- simRegData(nObs  = 100,
                       r2    = 0.5,
                       sigma = sigma,
                       beta  = matrix(c(0.25, rep(0.75, 10)))
                       )

## Specify 'sigma' as an inter-predictor correlation:
testData <- simRegData(nObs  = 100,
                       nVars = 10,
                       r2    = 0.5,
                       sigma = 0.2,
                       beta  = matrix(c(0.25, rep(0.75, 10)))
                       )
```

testImps *List of Example Data Elements*

Description

This data object is a list of data elements meant to demonstrate usage of the `plotImps` function.

Usage

```
data("testImps")
```

Format

A list with the following three elements.

impList: A list containing 10 multiply imputed datasets.

rMat: A logical pattern matrix flagging missing values in the data that were imputed to generate `impList`.

typeVec: A character vector giving the measurement level of each variable in `impList`.

See Also

plotImps

Index