Package: prova (via r-universe)

July 2, 2026

Title Nonparametric Probabilistic-Statistical Variate Analysis with Automated Markov-Chain Monte Carlo

Version 1.0.0

Description Calculate posterior joint and conditional probabilities, probability distributions of population frequencies, and information-theoretic measures, by means of Bayesian nonparametric methods. Data imputation is automatic and done in a principled way. Markov-chain Monte Carlo calculations are automatically handled and do not require user supervision. Applications range from statistical estimation and probabilistic hypothesis testing to evidence-based inference and decision making, in a wide range of disciplines from astrophysics to medicine. For more details and examples see for instance Porta Mana et al. (2026) <doi:10.31219/osf.io/8nr56>, Dunson & Bhattacharya (2011) <doi:10.1093/acprof:oso/9780199694587.003.0005>, Lindley & Novick (1981) <doi:10.1214/aos/1176345331>, Bernardo & Smith (2000) <doi:10.1002/9780470316870>, Müller et al. (2015) <doi:10.1007/978-3-319-18968-0>. Requires the packages 'Nimble', 'parallel', 'extraDistr'.

License AGPL (>= 3)

URL https://pglpm.github.io/prova/, https://github.com/pglpm/prova/

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.5.0)

Imports extraDistr, parallel

Suggests nimble (>= 1.4.2), knitr, rmarkdown

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

LazyData true

Repository https://pglpm.r-universe.dev

Date/Publication 2026-07-02 17:43:25 UTC

RemoteUrl https://github.com/pglpm/prova

RemoteRef HEAD

RemoteSha 93c004c9386a1b02a3efa44f29acb5dc6e7cd1bd

Contents

`flexiplot` *Plot numeric or character values*

Description

Plot function that modifies and expands the **graphics** package's `graphics::matplot()` function in several ways.

Usage

```
flexiplot(
  x,
  y,
  type = NULL,
  lty = c(1, 2, 4, 3, 6, 5),
  lwd = 2,
  pch = c(1, 2, 0, 5, 6, 3),
  col = palette(),
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  add = FALSE,
```

```
  xdomain = NULL,
  ydomain = NULL,
  alpha.f = 1,
  xjitter = NULL,
  yjitter = NULL,
  grid = TRUE,
  cex.main = 1,
  ...
)
```

Arguments

x	Numeric or character: vector of x-coordinates. If missing, a numeric vector 1:... is created having as many values as the rows of y.
y	Numeric or character: vector of y coordinates. If missing, a numeric vector 1:... is created having as many values as the rows of x.
type, lty, lwd, pch, col, xlab, ylab, add, cex.main	see analogous arguments in graphics::matplot().
xlim, ylim	NULL (default) or a vector of two values. In the latter case, if any of the two values is not finite (including NA or NULL), then the min or max x- or y-coordinates of the plotted points are used.
xdomain, ydomain	Character or numeric or NULL (default): vector of possible values of the variables represented in the x- and y-axes, in case the x or y argument is a character vector. The ordering of the values is respected. If NULL, then unique(x) or unique(y) is used.
alpha.f	Numeric, default 1: opacity of the colours, 0 being completely invisible and 1 completely opaque.
xjitter, yjitter	Logical or NULL (default): add base::jitter() to x- or y-values? Useful when plotting discrete variates. If NULL, jitter is added if the values are of character (or factor) class.
grid	Logical: whether to plot a light grid. Default TRUE.
...	Other parameters to be passed to graphics::matplot().

Details

This function is essentially a wrapper around graphics::matplot(), augmenting the latter with some additional features useful for plotting data and results handled by **Prova**. Some of the additional features provided by flexiplot are the following:

- Either or both x and y arguments can be of class base::character. In this case, axes labels corresponding to the unique values are used (see arguments xdomain and ydomain). This makes it easier to plot nominal and ordinal variates.
- A jitter can also be added to the generated points, via the xjitter and yjitter switches. This makes it easier to generate scatter plots of nominal and ordinal variates.

- It is possible to specify only a lower or upper limit in the `xlim` and `ylim` arguments, letting the other limit to be found automatically. This can be useful in plotting probabilities, in cases where we want to specify the lower, `0` limit, but want the upper limit to simply be the the maximum probability.
- Transparency of lines or markers can be specified through argument `alpha.f`.
- The plotting style is different, and default argument `type = 'l'` (line plot) rather than `type = 'p'` (point plot).

See the package's vignettes for more examples.

Value

`NULL`, invisibly; produces a plot, see `graphics::matplot()`.

See Also

`Pr()` to calculate posterior probabilities and quantiles.

`plot.probability()` to directly plot posterior probabilities and quantiles contained in a probability object.

`plotquantiles()` to plot quantile ranges.

Examples

```
## Scatter plot of the 'island' vs 'species' nominal variates of the penguins dataset;
## note how jitter is automatically added:
flexiplot(x = penguins[, 'species'], y = penguins[, 'island'])


## Scatter plot of the 'bill_len' vs 'species' variates of the penguins dataset:
flexiplot(x = penguins[, 'species'], y = penguins[, 'bill_len'])

## We can add jitter to separate the nominal values:
flexiplot(x = penguins[, 'species'], y = penguins[, 'bill_len'],
  xjitter = TRUE)


## Scatter plot of the 'bill_len' vs 'body_mass' variates;
## in this case we must specify the scatter-plot option `type = 'p'`:
flexiplot(x = penguins[, 'body_mass'], y = penguins[, 'bill_len'],
  type = 'p')

## Calculate the values of a normal distribution in a restricted range
x <- seq(from = -2, to = 2, length.out = 127)
y <- dnorm(x, mean = 0, sd = 1)

## plot the distribution, with 0 as the lower plot range:
flexiplot(x = x, y = y, ylim = c(0, NA))
```

`hist.probability` *Plot the variability of an object of class "probability" as a histogram*

Description

The posterior probabilities calculated with the `Pr()` function, and outputted as a `probability` object, have an associated variability that comes from the finite size of the data sample. This variability can be interpreted in two ways:

- How the probabilities would change, if we could collect a very large (infinite) amount of additional data, and how likely would such change be;
- The relative frequency of a particular variate value in the full (sampled and unsampled) population is unknown; we can quantify our uncertainty about this relative frequency with a probability distribution.

The `hist()` method for a `probability` object is a utility to visualize this kind of variability, in the form of a distribution.

Usage

```
## S3 method for class 'probability'
hist(
  x,
  subset = NULL,
  breaks = NULL,
  legend = "top",
  lty = c(1, 2, 4, 3, 6, 5),
  lwd = 2,
  col = palette(),
  alpha.f = 1,
  fill.alpha.f = 0.125,
  showmean = TRUE,
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = c(0, NA),
  main = NULL,
  grid = TRUE,
  add = FALSE,
  ...
)
```

Arguments

`x`	Object of class "probability", obtained with `Pr()`.
`subset`	Named list or named vector: which variate values to display. For the variates corresponding to the names in this list, only the vector of values corresponding to that variate is displayed.

breaks	NULL or as in function graphics::hist(). If NULL (default), an optimal number of breaks for each probability distribution is computed.
legend	One of the values "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center" (see graphics::legend()): plot a legend at that position. A value FALSE or any other does not plot any legend. Default "top".
lty, lwd, col, alpha.f, xlab, ylab, xlim, ylim, main, grid, add	see analogous arguments in graphics::matplot()
fill.alpha.f	Numeric, default 0.125: opacity of the histogram filling. 0 means no filling.
showmean	Logical, default TRUE: show the means of the probability distributions? The means correspond to the probabilities about the next observed unit.
...	Other parameters to be passed to flexiplot().

Value

Invisibly, an object of class "histogram".

See Also

Pr() to calculate posterior probabilities and quantiles.

plot.probability() to plot the posterior probabilities.

flexiplot() (on which hist.probability() is based) for more general plots.

plotquantiles() to plot quantile ranges.

Examples

```
## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## calculate the probability, and its variability,
## for the value 'Adelie' of the "species" variate
probs <- Pr(Y = data.frame(species = 'Adelie'), learnt = learnt, parallel = 1)
probs$values

## show the variability of this probability; equivalently show
## the probability distribution for the relative frequency of
## 'Adelie' penguins in the full population
hist(probs, legend = 'topright')
```

learn *Monte Carlo computation of posterior probability distribution*

Description

Compute the posterior joint probability distribution of the variates conditional on the given data, by means of Markov-chain Monte Carlo, using the package **Nimble**.

Usage

```
learn(
  data,
  metadata,
  auxdata = NULL,
  outputdir = NULL,
  nsamples = 3600,
  nchains = 8,
  nsamplesperchain = 450,
  parallel = TRUE,
  seed = NULL,
  cleanup = TRUE,
  appendinfo = TRUE,
  valueislearnt = TRUE,
  subsampledata = NULL,
  prior = missing(data) || is.null(data),
  startupMCiterations = 3600,
  minMCiterations = 0,
  maxMCiterations = +Inf,
  maxhours = +Inf,
  ncheckpoints = 12,
  maxrelMCSE = +Inf,
  minESS = 450,
  initES = 2,
  thinning = NULL,
  verbose = TRUE,
  plottraces = !cleanup,
  showKtraces = FALSE,
  showAlphatraces = FALSE,
 hyperparams = list(ncomponents = 64, minalpha = -4, maxalpha = 4, byalpha = 1, Rshapelo
  = 0.5, Rshapehi = 0.5, Rvarm1 = 3^2, Cshapelo = 0.5, Cshapehi = 0.5, Cvarm1 = 3^2,
  Dshapelo = 0.5, Dshapehi = 0.5, Dvarm1 = 3^2, Bshapelo = 1, Bshapehi = 1, Dthreshold
  = 1, tscalefactor = 4.266, Oprior = "Hadamard", Nprior = "Hadamard", avoidzeroW =
    NULL, initmethod = "datacentre", Qerror = pnorm(c(-1, 1)))
)
```

Arguments

data	A dataset, given as a base::data.frame() or as a file path to a CSV file. If missing or NULL, then the prior probability distribution is calculated.
metadata	metadata about the dataset's variates, given either as a data frame or as a file path to a CSV file.
auxdata	A larger dataset, given as a data frame or as a file path to a CSV file. Such a dataset would be too large to use in the Monte Carlo sampling, but can still be used to help estimate some hyperparameters.
outputdir	NULL (default) or NA or character: path to folder where output information and diagnostics should be saved. If NULL, a directory is created in the temporary-directory space given by base::tempdir(). If NA, a directory is created in the current working directory given by base::getwd(). If character, this is taken to be the output directory; it should of course be writable by the user.
nsamples	Integer, default 3600: number of desired, *approximately independent* Monte Carlo samples. If this argument is changed, the user is also required to explicitly give either nchains or nsamplesperchain, but not both; the remaining third argument is determined from $\text{nsamples} = \text{nchains} \times \text{nsamplesperchain}$.
nchains	Integer, default 8: number of Monte Carlo chains. If this argument is changed, the user is also required to explicitly give either nsamples or nsamplesperchain, but not both; the remaining third argument is determined from $\text{nsamples} = \text{nchains} \times \text{nsamplesperchain}$.
nsamplesperchain	Integer, default 450: number of *approximately independent* Monte Carlo samples per chain. If this argument is changed, the user is also required to explicitly give either nsamples or nchains, but not both; the remaining third argument is determined from $\text{nsamples} = \text{nchains} \times \text{nsamplesperchain}$.
parallel	Logical or positive integer or cluster object. TRUE (default): use roughly half of available cores; FALSE (default): use serial computation; integer: use this many cores. It can also be a cluster object previously created with parallel::makeCluster(); in this case the parallel computation will use this object.
seed	Integer or NULL (default): use this seed for the random number generator. If NULL, do not set the seed.
cleanup	Logical, default TRUE: remove diagnostic files at the end of the computation?
appendinfo	Logical, default TRUE: append information about number of variates ('V'), number of data points ('D'), number of Monte Carlo samples ('S'), and timestamp, to the name of the output directory outputdir? The appended string has the format 'Vn_Dn_Sn_YYMMDDTHHMMSS'.
valueislearnt	Logical or NULL: should the VALUE returned be the learnt object containing the results from the Monte Carlo computation? Default TRUE. If FALSE, then VALUE is the output directory name. If NULL, then VALUE is NULL.
subsampledata	Integer or NULL (default): if integer, use only that many datapoints from the original dataset in the data argument.
prior	Logical: Calculate the prior distribution? Default is FALSE unless data argument is missing or NULL.

`startupMCiterations`
: Integer, default 3600: number of initial Monte Carlo iterations.

`minMCiterations`
: Integer, default 0: minimum number of Monte Carlo iterations to be doneby a chain.

`maxMCiterations`
: Integer, default `Inf`: Do at most this many Monte Carlo iterations per chain.

`maxhours`
: Numeric, default `Inf`: approximate time limit, in hours, for the Monte Carlo computation to last.

`ncheckpoints`
: Integer or `NULL`, default 12: number of datapoints (per chain) to use for checking when the Monte Carlo computation should end. If `NULL`, this is equal to number of variates + 2. If Inf, use all datapoints.

`maxrelMCSE`
: Numeric positive, default `+Inf`: desired maximal *relative Monte Carlo Standard Error* of calculated probabilities with respect to their variability with new data. The default `+Inf` means that `minESS` is used instead. `maxrelMCSE` is related to `minESS` by $\mathrm{maxrelMCSE} = 1/\sqrt{\mathrm{minESS} + \mathrm{initES}}$.

`minESS`
: Numeric positive or `NULL`, default 450: desired minimal Monte Carlo *Expected Sample Size*. If `NULL`, it is equal to the final `nsamplesperchain`. `minESS` is related to `maxrelMCSE` by $\mathrm{minESS} = 1/\mathrm{maxrelMCSE}^2 - \mathrm{initES}$.

`initES`
: Numeric positive, default 2: number of initial "burn-in" samples, separated by the Expected Sample Size, to be discarded. Note that the Monte Carlo chain typically starts in a high-probability region, so there is no reason to discard many initial samples.

`thinning`
: Integer or `NULL` (default): thin out the Monte Carlo samples by this value. If `NULL`: let the diagnostics decide the thinning value.

`verbose`
: Logical, default `TRUE`: output the progress to terminal? If `FALSE`, the progress is outputted to the file `'main.log'` in the `outputdir` directory.

`plottraces`
: Logical, default `TRUE`: save plots of the Monte Carlo traces of diagnostic values?

`showKtraces`
: Logical, default `FALSE`: save plots of the Monte Carlo traces of the K parameter?

`showAlphatraces`
: Logical, default `FALSE`: save plots of the Monte Carlo traces of the Alpha parameter?

`hyperparams`
: List: hyperparameters of the hyperprior; see values in "Usage".

Details

This function takes as main inputs a set of data and metadata, and computes the full joint probability distribution for new data, including its variability. From this full joint distribution any other distributions of interest can subsequently be computed; see `Pr()` and related functions. This computation can also be interpreted as an estimation of the full joint frequency distribution of the variates in the *whole population*, beyond the sample data, together with its uncertainty. The computation allows for the use of datapoints with partially missing variables: imputation is automatically made. This imputation is *principled*, made according to the rules of probability theory.

The output is a "learnt" object, typically saved in a `learnt.rds` file, which is used in all subsequent probabilistic computations. Other information about the computation is provided in logs and plots, saved in a directory specified by the user.

See `vignette('intro')` for introductory examples.

The computation is "non-parametric": probability or frequency distributions are not assumed to be Gaussian or of any other specific shape; no "model" is assumed. The mathematical representation of the space of joint frequency distributions follows ideas of Dunson & Bhattacharya (2011); see technical manual for details.

The computation is done via Markov-chain Monte Carlo, using the package **Nimble**. "Convergence" of the Monte Carlo computation is automatically assessed with methods described in Vehtari & al. (2021) and Kwon & al. (2025); see technical manual for details. The default values for convergence require that all of the following three conditions be fulilled:

- The computation's numerical error (Monte-Carlo Standard Error) for the posterior probability must be smaller than 4.7% of the standard deviation of the posterior's variability.
- The computation's numerical error for the 0.055- and 0.945-percentiles of the posterior's variability should be smaller than 4.7% of the distance between them.

Typically this requirement leads to final results obtained with the `Pr()` function having at least two significant digits.

The `learn()` function can take hours or even days to perform its computations, depending on the size of the dataset, number of variates, and the (initially unknown) "shape" of the underlying probability distribution. For this reason it is typically called within an R script, executed via utils::Rscript. For example, a script 'myscript.R' could have the following structure:

```
library('prova')

learn(
  data = 'filename_with_data.csv', # CSV file containing the dataset
  metadata = 'filename_with_metadata.csv', # CSV file containing the metadata
  outputdir = 'some_directory', # path to output directory
  parallel = 8 # machine has more than 8 cores, so we use 8
  ## possibly other arguments to learn()
)
```

and then be called on a bash terminal with

```
$ Rscript myscript.R > learnoutput.log 2>&1 &
```

with such a call, the file 'learnoutput.log' will contain information about how the computation is proceeding and the estimated end time.

Value

A "learnt" object, or name of directory containing such an object and other output files, or `NULL`, depending on argument `valueislearnt`.

`learn()` saves several files in a directory. By default this output directory is a temporary directory within the one used by `base::tempdir()`, but an alternative one can be chosen with the argument `outputdir =`. The output directory contain several diagnostic files for the Monte Carlo computation; in particular:

- `MCtraces.pdf`: shows several trace plots of the Monte Carlo sampling; the correspondin data are in the file `MCtraces.rds`.
- `plotsamples_learnt.pdf`, `plotquantiles_learnt.pdf`: show the marginal posterior distributions of each individual variate, together with their variability (as samples or quantiles).
- `log-1.log`, `log-2.log`, ... one for each parallel core; report the progress of each parallel Monte Carlo computation and notes about it.
- `rng_seed.rds`: the state of the pseudorandom seed (see base::Random) when `learn()` was called.
- `metadata.csv`: a copy of the metadata.

It is recommended that you give an explicit argument `outputdir =` and save the directory with the files above for future reference. In particular, the `MCtraces.pdf` plot and `MCtraces.rds` data can be useful to report Monte Carlo convergence in any work of yours that used **Prova**.

References

For the mathematical representation of the frequency space:

- Dunson, Bhattacharya (2011): *Nonparametric Bayes regression and classification through mixtures of product kernels* doi:10.1093/acprof:oso/9780199694587.003.0005.
- Ishwaran, Zarepour (2002): *Exact and approximate sum representations for the Dirichlet process* doi:10.2307/3315951.
- Porta Mana https://github.com/pglpm/prova/raw/main/development/manual/pglpm2024-bayes_nonparam.pdf.

About Bayesian inference under exchangeability ("population inference"):

- Lindley, Novick (1981): *The role of exchangeability in inference*, doi:10.1214/aos/1176345331.
- Bernardo, Smith (2000): *Bayesian Theory*. Wiley doi:10.1002/9780470316870.
- Porta Mana https://github.com/pglpm/prova/raw/main/development/manual/pglpm2024-bayes_nonparam.pdf.

About nonparametrics:

- Müller et al. (2015): *Nonparametric Bayesian inference*. IMS doi:10.1007/978-3-319-18968-0.
- Hjort et al. (2010): *Bayesian Nonparametrics*. Cambridge University Press doi:10.1017/CBO9780511802478.

About Markov-chain Monte Carlo and "convergence":

- de Valpine, Paciorek, Turek, & al. (2026): *NIMBLE: MCMC, Particle Filtering, and Programmable Hierarchical Modeling* doi:10.5281/zenodo.1211190, https://cran.r-project.org/package=nimble.
- Kwon & al. (2025): *MCMC stopping rules in latent variable modelling* doi:10.1111/bmsp.12357.
- Vehtari & al. (2021): *Rank-normalization, folding, and localization: an improved R-hat for assessing convergence of MCMC* doi:10.1214/20-BA1221.
- Roy (2020): *Convergence diagnostics for Markov chain Monte Carlo* doi:10.1146/annurev-statistics-031219-04

- Gilks & al. (1998): *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC doi:10.1201/b14835.
- D. J. C. MacKay (2005): *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press https://www.inference.org.uk/itila/book.html.
- Porta Mana https://github.com/pglpm/prova/raw/main/development/manual/pglpm2024-bayes_nonparam.pdf.

See Also

`metadatatemplate()` to help writing metadata files.

`Pr()` to calculate probabilities, and `qPr()` to calculate quantiles, given the data processed by `learn()`.

`rPr()` to generate datapoints similar to the data processed by `learn()`.

`mutualinfo()` to calculate mutual information given the data processed by `learn()`.

`pread.csv()` and `pwrite.csv()` to read and write CSV files in the format used by `learn()`.

Examples

```
### WARNING: the following example, if run, might even take a minute or more.

## Create dataset with 3 points of variate 'V' for demonstration:
dataset <- data.frame(V = rnorm(n = 3))

## Create metadata file:
metadata <- data.frame(name = 'V', type = 'continuous')

## Learn from the data:
learnt <- learn(
  data = dataset, metadata = metadata,
      ## the following parameters are unrealistic
      ## only used to reduce computation time for this example
  nsamples = 10, nchains = 1,
  startupMCiterations = 10, maxMCiterations = 10,
  minESS = 0, initES = 0
)

## Check structure of `learnt` object:
str(learnt)
```

`learntExample` *Example* `learnt` *object produced by learn()*

Description

An example `learnt` object obtained by means of the `learn()` function, using the datasets::penguins dataset and the metadata in metadataExample, according to the call

```
learn(data = penguins, metadata = metadataExample,
  nsamples = 225, nchains = 15)
```

It is a list that essentially contains posterior hyperparameters for drawing statistical inferences about the variates `species` and `bill_len`.

Note that the `learn()` function that produced `learntExample` was called with the option to create only a limited number (225) of Monte Carlo samples, in order to reduce its memory size. Thus the numerical error associated with the Monte Carlo approximation is relatively in inferences drawn from the posterior hyperparameters saved in `learntExample`. It is only meant to be used for illustration purposes of the package's capabilities.

Usage

```
learntExample
```

Format

`learntExample`**:**

A list containing results from Markov-chain Monte Carlo computation, including diagnostics and variate metadata.

Value

No return value.

See Also

`learn()`, which produces this kind of object.

`Pr()`, `qPr()`, `rPr()`, `mutualinfo()`: functions that require this kind of object in order to calculate probabilities and quantiles, generate data points, and calculate mutual information.

`metadataExample` *Example metadata file*

Description

A data frame containing the prior information about the variates `species` and `bill_len` of the datasets::penguins dataset.

Usage

```
metadataExample
```

Format

`metadataExample`**:**

A data frame with 2 rows and 10 columns.

Value

No return value.

See Also

`metadatatemplate()` which helps producing this kind of metadata files from a given dataset.

`learn()` which needs this kind of metadata files to "learn" from data.

`metadatatemplate` *Metadata and helper function for metadata*

Description

Metadata and helper function to create a template metadata file or object.

Usage

```
metadatatemplate(
  data,
  file = NULL,
  includevrt = NULL,
  excludevrt = NULL,
  addsummary2metadata = FALSE,
  backupfiles = FALSE,
  verbose = TRUE
)
```

Arguments

`data`	A dataset, given as a data frame or as a file path to a csv file.
`file`	Character or `NULL` (default): name of csv file where the metadata should be saved; if `NULL`: output metadata as `VALUE`.
`includevrt`	Character or `NULL`: name of variates in dataset to be included.
`excludevrt`	Character or `NULL`: name of variates in dataset to be excluded.
`addsummary2metadata`	Logical: also output some diagnostic statistics in the metadata? Default `FALSE`.
`backupfiles`	Logical: rename previous metadata file if it exists? Default `TRUE`.
`verbose`	Logical: output heuristics for each variate? Default `TRUE`.

Details

The learn() function needs metadata about the variates present in the data. Such metadata can be provided either as a csv file or as a base::data.frame(). The function buildmetadata creates a template metadata csv-file, or outputs a metadata data.frame, by trying to *guess* metadata information from the dataset.The guesses may be very incorrect (as already said, metadata is information not contained in the data, so no algorithm can exist that extracts it from the data). **The user *must* modify and correct this template, using it as a starting point to prepare the correct metadata information.**

Value

A preliminary data frame containing the metadata, invisibly if file = NULL. If argument file is a character, a preliminary metadata file is also created with that name or path.

Metadata information and format

In order to correctly learn from a dataset, the learn() function needs information that is not contained in the data themeselves; that is, it needs *meta*data. Metadata are provided either as a csv file or as a base::data.frame().

A metadata file or data.frame must contain one row for each simple variate in the given inference problem, and the following fields (columns), even if some of them may be empty:

name, type, domainmin, domainmax, datastep, minincluded, maxincluded, V1, V2, (possibly additional V-fields, sequentially numbered)

The type field has three possible values: nominal, ordinal, continuous. The remaining fields that must be filled in depend on the type field. Here is a list of requirements:

- nominal and ordinal: require *either* V1, V2, ... fields *or* domainmin, domainmax, datastep (all three) fields. No other fields are required.
- continuous: requires domainmin, domainmax, datastep, minincluded, maxincluded.

Here are the meanings and possible values of the fields:

name: The name of the variate. This must be the same character string as it appears in the dataset (be careful about upper- and lower-case).

type: The data type of variate name. Possible values are nominal, ordinal, continuous.

- A *nominal* (also called *categorical*) variate has a discrete, finite number of possible values which have no intrinsic ordering. Examples could be a variate related to colour, with values "red", "green", "blue", and so on; or a variate related to cat breeds, with values "Siamese", "Abyssinian", "Persian", and so on. The possible values of the variate must be given in the fields V1, V2, and so on. It is important to include values that are possible but are *not* present in the dataset. A variate having only two possible values (binary variate), for example "yes" and "no", can be specified as nominal.
- An *ordinal* variate has a discrete, finite number of possible values which do have an intrinsic ordering. Examples could be a Likert-scaled variate for the results of a survey, with values "very dissatisfied", "dissatisfied", "satisfied", "very satisfied"; or a variate related to the levels of some quantities, with values "low", "medium", "high"; or a variate having a numeric scale with values from 1 to 10. Whether a variate is nominal or ordinal often depends on

the context. The possible values of the variate but be given in either one (but not both) or two ways: (1) in the fields `V1`, `V2`, ..., as for nominal variates; (2) as the fields `domainmin`, `domainmax`, `datastep`. Option (2) only works with numeric, equally spaced values: it assumes that the first value is `domainmin`, the second is `domainmin+datastep`, the third is `domainmin+2*datastep`, and so on up to the last value, `domainmax`.

- A *continuous* variate has a continuum of values with an intrinsic ordering. Examples could be a variate related to the width of an object; or to the age of a person; or one coordinate of an object in a particular reference system. A continuous variate requires specification of the fields `domainmin`, `domainmax`, `datastep`, `minincluded`, `maxincluded`. Some naturally continuous variates are often rounded to a given precision; for instance, the age of a person might be reported as rounded to the nearest year (25 years, 26 years, and so on); or the length of an object might be reported to the nearest centimetre (1 m, 1.01 m, 1.02 m, and so on). The minimum distance between such rounded values **must** be reported in the `datastep` field; this would be `1` in the age example and `0.01` in the length example above. See below for further explanation of why reporting such rounding is important.

`domainmin`: The minimum value that the variate (ordinal or continuous) can take on. Possible values are a real number or an empty value, which is then interpreted as `-Inf` (explicit values like `-Inf`, `-inf`, `-infinity` should also work). Some continuous variates, like age or distance or temperature, are naturally positive, and therefore have `domainmin` equal `0`. But in other contexts the minimum value could be different. For instance, if a given inference problem only involves people of age 18 or more, then `domainmin` would be set to `18`.

`domainmax`: The maximum value that the variate (ordinal or continuous) can take on. Possible values are a real number, or an empty value, which is then interpreted as `+Inf` (explicit values like `Inf`, `inf`, `infinity` should also work). As with `domainmin`, the maximum value depends on the context. An age-related variate could theoretically have `domainmax` equal to infinity (empty value in the metadata file); but if a given study categorizes some people as "90 years old or older", then `domainmax` should be set to `90`.

`datastep`: The minimum distance between the values of a variate (ordinal or continuous). Possible values are a positive real number or an empty value, which is then interpreted as 0 (the explicit value `0` is also accepted). For a numeric ordinal variate, `datastep` is the step between consecutive values. For a continuous *rounded* variate, `datastep` is the minimum distance between different values that occurs because of rounding; see the examples given above. The function `buildmetadata` has some heuristics to determine whether the variate is rounded or not. See further details under the section Rounding below.

`minincluded`, `maxincluded`: Whether the minimum (`domainmin`) and maximum(`domainmax`) values of a *continuous* variate can really appear in the data or not. Possible values are `true` (or `t` or `yes`) or `false` (or `f`, `no`, or an empty field); upper- or lower-case is irrelevant. Here are some examples about the meaning of these fields. (a) A continuous *unrounded* variate such as temperature has 0 as a minimum possible value `domainmin`, but this value itself is physically impossible and can never appear in data; in this case `minincluded` is empty (or set to `false` or `no`). (b) A variate related to the *unrounded* length, in metres, of some objects may take on any positive real value; but suppose that all objects of length 5 or less are grouped together under the value `5`. It is then possible for several datapoints to have value `5`: one such datapoint could originally have the value 3.782341...; another the value 4.929673..., and so on. In this case `domainmin` is set to `5`, and `minincluded` is set to `true` (or `yes`). Similarly for the maximum value of a variate and `maxincluded`. Note that if `domainmin` is minus-infinity (empty value in the metadata file), then `minincluded` is automatically empty (that is, `false`), and similarly for `maxincluded` if `domainmax` is infinity.

See Also

learn(), which generates the information necessary to calculate posterior probabilities, based on data and metadata.

Examples

```
## Create a preliminary data frame of metadata for the `penguins` dataset
metadata <- metadatatemplate(data = datasets::penguins, file = NULL)

## Note how the preliminary data frame includes additional spots
## for values of nominal and ordinal variates
## which could be missing from the data
print(metadata)


## Create a preliminary data frame of metadata for the `penguins` dataset,
## including only the 'species' and 'bill_len' variates:
metadata2 <- metadatatemplate(
  data = datasets::penguins, file = NULL,
  includevrt = c('species', 'bill_len')
)

print(metadata2)


## Create a preliminary data frame of metadata for the `penguins` dataset,
## excluding the 'year' variate:
metadata3 <- metadatatemplate(
  data = datasets::penguins, file = NULL,
  excludevrt = 'year'
)

print(metadata3)

## Generate 10 points for a continuous variate in (0, 1)
dataset <- runif(10)

## `metadatatemplate` correctly guesses the variate minimum,
## but not the maximum (`NA` is equivalent to `+Inf`)
metadata <- metadatatemplate(data = dataset, file = NULL)
print(metadata)
```

---

mutualinfo *Calculate mutual information between groups of joint variates*

Description

This function calculates various entropic information measures between two grops of joint variates: the mutual information, the conditional entropies, and the entropies.

Usage

```
mutualinfo(
  Y1names,
  Y2names = NULL,
  X = NULL,
  learnt,
  tails = NULL,
  n = NULL,
  unit = "Sh",
  parallel = TRUE,
  verbose = FALSE
)
```

Arguments

Y1names	Character vector: first group of joint variates
Y2names	Character vector or NULL: second group of joint variates
X	Matrix or data.frame or NULL: values of some variates conditional on which we want the probabilities.
learnt	Either a character with the name of a directory or full path for an 'learnt.rds' object, or such an object itself.
tails	Named vector or list, or NULL (default). The names must match some or all of the variates in arguments X. For variates in this list, the probability conditional is understood in an semi-open interval sense: $X \leq x$ or $X \geq x$, an so on. See analogous argument in Pr().
n	Integer or NULL (default): number of samples from which to approximately calculate the mutual information. Default as many as Monte Carlo samples in learnt.
unit	Either one of 'Sh' for *shannon* (default), 'Hart' for *hartley*, 'nat' for *natural unit*, or a positive real indicating the base of the logarithms to be used.
parallel	Logical or positive integer or cluster object. TRUE (default): use roughly half of available cores; FALSE: use serial computation; integer: use this many cores. It can also be a cluster object previously created with parallel::makeCluster(); in this case the parallel computation will use this object.
verbose	Logical, default FALSE: give messages about parallel processing?

Details

If Y_1 and Y_2 are two variates, each of which can be a joint variate such as $Y_1 = (Y_{1,1}, Y_{1,2}, \dots)$, and X a third, also possibly join, variate, then the mutual information MI between Y_1 and Y_2, conditional on $X = x$, is given by

$$MI(Y_1, Y_2|X = x) := \sum_{y_1, y_2} \Pr(Y_1 = y_1, Y_2 = y_2|X = x, \text{data}) \log_2 \frac{\Pr(Y_1 = y_1, Y_2 = y_2|X = x, \text{data})}{\Pr(Y_1 = y_1|X = x, \text{data}) \cdot \Pr(Y_2 = y_2|X = x, \text{data})}$$

an expression which can also be written in several other equivalent ways. It is an information-theoretic measure of association that is model-free, that is, does not depend on assumptions such as

linearity, gaussianity, and similar. See `vignette('mutualinfo')` for discussion and example uses, and also the "References" section. If Y_1, Y_2 are *jointly gaussian variates*, then there is a mathematical correspondence between their mutual information and their Pearson correlation coefficient; see output `MI.rGauss` in the "Value" section.

The conditional entropy of Y_1 with respect to Y_2, conditional on $X = x$, is given by

$$\mathit{CondEn12}(Y_1, Y_2|X = x) := -\sum_{y_1,y_2} \Pr(Y_1 = y_1|Y_2 = y_2, X = x, \text{data}) \log_2 \Pr(Y_1 = y_1|Y_2 = y_2, X = x, \text{data}) \cdot \Pr(Y_2 =$$

The (differential) entropy of Y_1, conditional on $X = x$, is given by

$$\mathit{En1}(Y_1|X = x) := -\sum_{y_1} \Pr(Y_1 = y_1|X = x, \text{data}) \log_2 \Pr(Y_1 = y_1|X = x, \text{data}) \,\mathrm{Sh}$$

see "References" section for discussions about entropy and conditional entropy.

The function `mutualinfo()` calculates the quantities above for the joint variates specified in the arguments `Y1names` and `Y2names`, conditional on the values of the variates specified in the data frame `X`. If `X` is omitted or `NULL`, then the posterior probabilities $\Pr(Y_1|\text{data})$ etc. are used. Each variate in the argument `X` can be specified either as a point-value $X = x$ or as a left-open interval $X \leq x$ or as a right-open interval $X \geq x$, through the argument `tails`.

The computation of these quantities is done via Monte Carlo integration, using the samples produced by the `learn()` function. The present function also output the numerical error associated with this computation.

Value

A list consisting of the following elements:

- `MI`, a vector of `value` and `accuracy`: the mutual information between (joint) variates `Y1names` and (joint) variates `Y2names`.
- `CondEn12`, `CondEn21`, vectors of `value` and `accuracy`: the conditional entropy of the first variate given the second, and vice versa.
- `En1`, `En2`, vectors of `value` and `accuracy`: the (differential) entropies of the first and second variates.
- `MI.rGauss`, a vector of `value` and `accuracy`: the absolute value of the Pearson correlation coefficient r of a *multivariate Gaussian distribution* having mutual information `MI`; the two are related by $\mathrm{MI} = -\ln(1 - r^2)/2$. It may provide a vague intuition for the `MI` value for people more familiar with Pearson's correlation, but should be taken with a grain of salt.
- `unit`, `Y1names`, `Y1names`: same as the input arguments, included for the user's convenience.

See Also

`Pr()` to calculate probabilities and their variability.

`learn()`, which generates the `learnt` objects required by `mutualinfo()`.

Examples

```
## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## mutual information between variates 'species' and 'bill_len'
MI <- mutualinfo(Y1names = 'species', Y2names = 'bill_len',
  learnt = learnt, parallel = 1)

paste0(MI$MI, ' ', MI$unit, collapse = ' +/- ')

## Shannon entropy of variate 'species'
paste0(MI$En1, ' ', MI$unit, collapse = ' +/- ')



## Shannon entropy of variate 'species',
## conditional on a bill length of 30 mm:
entr <- mutualinfo(
  Y1names = 'species',
  X = data.frame(bill_len = 30),
  learnt = learnt, parallel = 1
)

paste0(entr$En1, ' ', entr$unit, collapse = ' +/- ')

## the entropy is now lower; indeed a penguin with a short bill length
## is most probably of the 'Adelie' species:
probs <- Pr(
  Y = data.frame(species = c('Adelie', 'Gentoo', 'Chinstrap')),
  X = data.frame(bill_len = 30),
  learnt = learnt, parallel = 1
)

print(probs)
```

`plot.probability` *Plot an object of class "probability"*

Description

This `base::plot()` method is a utility to plot probabilities obtained with `Pr()`, as well as their variabilities. The probabilities are plotted either against `Y`, with one curve for each value of `X`, or vice versa.

Usage

```
## S3 method for class 'probability'
plot(
  x,
  variability = NULL,
  subset = NULL,
  PvsY = NULL,
  legend = "top",
  lty = c(1, 2, 4, 3, 6, 5),
  lwd = 2,
  col = palette(),
  type = NULL,
  alpha.f = 1,
  var.alpha.f = NULL,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  ylim = c(0, NA),
  grid = TRUE,
  add = FALSE,
  ...
)
```

Arguments

x	Object of class "probability", obtained with Pr().
variability	One of the values 'quantiles', 'samples', 'none' (equivalent to NA or FALSE), or NULL (default), in which case the variability available in p is used. This argument chooses how to represent the variability of the probability; see Pr(). If the requested variability is not available in the object p, then a warning is issued and no variability is plotted.
subset	Named list or named vector: which variate values to display. For the variates corresponding to the names in this list, only the vector of values corresponding to that variate is displayed.
PvsY	Logical or NULL: should probabilities be plotted against their Y argument? If NULL, the argument between Y and X having larger number of values is chosen. As many probability curves will be plotted as the number of values of the other argument.
legend	One of the values 'bottomright', 'bottom', 'bottomleft', 'left', 'topleft', 'top', 'topright', 'right', 'center' (see graphics::legend()): plot a legend at that position. A value FALSE or any other does not plot any legend. Default 'top'.
lty, lwd, col, type, xlab, ylab, main, ylim, grid, add	see analogous arguments in graphics::matplot()
alpha.f	Numeric, default 0.25: opacity of the colours, 0 being completely invisible and 1 completely opaque.

var.alpha.f	Numeric: opacity of the quantile bands or of the samples, 0 being completely invisible and 1 completely opaque.
...	Other parameters to be passed to flexiplot().

Value

NULL, invisibly; produces a plot, see graphics::matplot().

See Also

Pr() to calculate posterior probabilities and quantiles.

hist.probability() to plot the variability of the probabilities as a distribution.

flexiplot() (on which plot.probability() is based) for more general plots.

plotquantiles() to plot quantile ranges.

Examples

```
## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## create a grid of values for variate "bill length",
## based on the information in the dataset and metadata:
values <- vrtgrid(vrt = 'bill_len', learnt = learnt)

## calculate the probabilities and quantiles
probs <- Pr(Y = data.frame(bill_len = values), learnt = learnt, parallel = 1)

## plot the probabilities and quantiles
plot(probs)
```

---

plotquantiles *Plot pairs of quantiles*

Description

Utility function to plot pairs of quantiles obtained with Pr().

Usage

```
plotquantiles(
  x,
  y,
  xdomain = NULL,
  alpha.f = 0.25,
  col = palette(),
```

```
  border = NA,
  type = "n",
  ...
)
```

Arguments

x	Numeric or character: vector of x-coordinates. See `flexiplot()`.
y	Numeric: a matrix having as many rows as `x` and an even number of columns, with one column per quantile. Typically these quantiles have been obtained with `Pr()`, as their `$quantiles` value. This value is a three-dimensional array, and one of its columns (corresponding to the possible values of the `X` argument of `Pr()`) or one of its rows (corresponding to the possible values of the `Y` argument of `Pr()`) should be selected before being used as `y` input.
xdomain	Character or numeric or `NULL` (default): vector of possible values of the variable represented in the x-axis, if the `x` argument is a character vector. The ordering of the values is respected. If `NULL`, then `unique(x)` is used.
alpha.f	Numeric, default 0.25: opacity of the quantile bands, `0` being completely invisible and `1` completely opaque.
col	Fill colour of the quantile bands. Can be specified in any of the usual ways, see for instance `grDevices::col2rgb()`. Default `#4477AA`.
border	Fill colour of the quantile bands. Can be specified in any of the usual ways, see for instance `grDevices::col2rgb()`. If `NA` (default), no border is drawn.
type	see analogous argument in `flexiplot()`.
...	Other parameters to be passed to `flexiplot()`.

Value

`NULL`, invisibly; produces a plot, see `graphics::matplot()`.

See Also

`Pr()` to calculate posterior probabilities and quantiles.

`plot.probability()` to directly plot posterior probabilities and quantiles contained in a probability object.

`flexiplot()` for more general plots.

Examples

```
## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## create a grid of values for variate "bill length",
## based on the information in the dataset and metadata:
values <- vrtgrid(vrt = 'bill_len', learnt = learnt)
```

```
## calculate the probabilities and quantiles
probs <- Pr(Y = data.frame(bill_len = values), learnt = learnt, parallel = 1)

## plot the quantiles, setting lower plot range to zero
plotquantiles(x = values, y = probs$quantiles[, 1, ], ylim = c(0, NA),
  xlab = 'bill length', ylab = 'probability')

## add a plot of the probabilities in thick dashed red
flexiplot(x = values, y = probs$values, lwd = 5, lty = 2, col = 2, add = TRUE)
```

`Pr` *Calculate posterior probabilities*

Description

This function calculates posterior probability densities, cumulative posterior probabilities, and mixtures thereof. It also outputs the variability of such probabilities if more training data were available, and the Monte Carlo Standard Error for the calculated posterior probabilities.

Usage

```
Pr(
  Y,
  X = NULL,
  learnt,
  tails = NULL,
  priorY = NULL,
  nsamples = "all",
  quantiles = c(0.055, 0.25, 0.75, 0.945),
  parallel = TRUE,
  sep = ",",
  solidus = "|",
  verbose = FALSE,
  keepYX = TRUE
)
```

Arguments

`Y`	Matrix or data.table: set of values of variates of which we want the joint probability of. One variate per column, one set of values per row.
`X`	Matrix or data.table or `NULL` (default): set of values of variates on which we want to condition the joint probability of `Y`. If `NULL`, no conditioning is made (except for conditioning on the learning dataset and prior assumptions). One variate per column, one set of values per row.
`learnt`	Either a character with the name of a directory or full path for a 'learnt.rds' object, produced by the `learn()` function, or such an object itself.

`tails` Named vector or list, or `NULL` (default). The names must match some or all of the variates in arguments `Y` and `X`. For variates in this list, the probability arguments are understood in an semi-open interval sense: $Y \leq y$ or $Y \geq y$, an so on. This is true for `Y` and `X` variates (on the left and on the right of the conditional sign `|`). A left-open interval $Y \leq y$ is indicated by `'<='` or `'left'` or `-1`; a right-open interval $Y \geq y$ is indicated by `'>='` or `'right'` or `+1`. Values `NULL`, `'=='`, `0` indicate that a point value `Y = y` (not an interval) should be calculated. **NB**: the semi-open intervals *always* include the given value; this is important for ordinal or rounded variates. For instance, if Y is an integer variate, then to calculate $\Pr(Y < 3)$ you should require $\Pr(Y \leq 2)$; for this reason we also have that $\Pr(Y \leq 2)$ and $\Pr(Y \geq 2)$ generally add up to *more* than 1.

`priorY` Numeric vector with the same length as the rows of `Y`, or `TRUE`, or `NULL` (default): prior probabilities or base rates for the `Y` values. If `TRUE`, the prior probabilities are assumed to be all equal.

`nsamples` Integer or `NULL` or `'all'` (default): desired number of samples of the variability of the probability for `Y`. If `NULL`, no samples are reported. If `'all'` (or `Inf`), all samples obtained by the `learn()` function are used.

`quantiles` Numeric vector, between 0 and 1, or `NULL`: desired quantiles of the variability of the probability for `Y`. Default `c(0.055, 0.25, 0.75, 0.945)`, that is, the 5.5%, 25%, 75%, 94.5% quantiles. These are typical quantile values in the Bayesian literature: they give 50% and 89% credibility intervals, which correspond to 1 shannons and 0.5 shannons of uncertainty (see doi:10.5281/zenodo.17072199). If `NULL`, no quantiles are calculated.

`parallel` Logical or positive integer or cluster object. `TRUE` (default): use roughly half of available cores; `FALSE`: use serial computation; integer: use this many cores. It can also be a cluster object previously created with `parallel::makeCluster()`; in this case the parallel computation will use this object.

`sep` character, default `','`: character to separate variate names and values

`solidus` character, default `'|'`: character prepended to names of the variates in the conditional (typically the `X` variates).

`verbose` Logical, default `FALSE`: give messages about parallel processing?

`keepYX` Logical, default `TRUE`: keep a copy of the `Y` and `X` arguments in the output? This is used for the plot method.

Details

This function calculates the posterior probability $\Pr(Y = y|X = x, \text{data})$, where $Y = y$ and $X = x$ are two (non overlapping) sets of joint variate values, inputted as data frame arguments `Y` and `X`. It is somewhat analogous to the d-variants and p-variantes of R distribution functions, such as `stats::dnorm()` and `stats::pnorm()`. If `X` is omitted or `NULL`, then the posterior probability $\Pr(Y = y|\text{data})$ is calculated.

For some variates in `Y` or `X`, tail values can also be prescribed, so that this function calculates mixed probabilities such as

$$\Pr(Y_1 = y_1, Y_2 \leq y_2, \ldots | X_1 = x_1, X_2 \geq x_2, \ldots, \text{data}) \ .$$

Tail values are inputted via the `'tails'` argument; see "Usage".

This function also outputs the variability of the posterior probabilities above, that is, probabilities such as $\Pr(Y = y|X = x, \text{new data}, \text{data})$ that we could have if more learning data were provided, as well as a number of samples of the possible values of such probability. This variability can be outputted in two ways; the user can choose either, or both, or none:

- As samples (default 3600 samples, depending on the 'nsamples' argument given to the `learn()` function) of the alternative values that the posterior probability could have.
- As quantiles (default 5.5%, 25%, 75%, 94.5%) of the possible variability.

If several joint values are given for `Y` or `X`, the function will create a 2D grid of results for all possible combinations of the given `Y` and `X` values.

This function also allows for base-rate or other prior-probability corrections: If a prior (for instance, a base rate) for `Y` is given, the function will calculate the probability $\Pr(Y = y|X = x, \text{data}, \text{prior})$ from $\Pr(X = x|Y = y, \text{data})$ and the prior, by means of Bayes's theorem.

Each variate in each argument `Y`, `X` can be specified either as a point-value $Y = y$ or as a left-open interval $Y \leq y$ or as a right-open interval $Y \geq y$, through the argument `tails`.

See `vignette('intro')` for example uses.

Value

An object of class "probability", effectively a list consisting of the following elements:

- `values`: a matrix with the probabilities $\Pr(Y = y|X = x, \text{data})$, for all joint values y of the Y-variates (rows) and all joint values x of the X-variates (columns).
- `quantiles` (possibly `NULL`): an array with the variability quantiles (3rd dimension of the array) for such probabilities.
- `samples` (possibly `NULL`): an array with the variability samples (3rd dimension of the array) for such probabilities.
- `values.MCaccuracy`, `quantiles.MCaccuracy`: arrays with the numerical accuracies (roughly speaking a standard deviation) of the Monte Carlo calculations for the `values` and `quantiles` elements.
- `Y`, `X`: copies of the `Y` and `X` arguments.

References

- Lindley, Novick (1981): *The role of exchangeability in inference*, doi:10.1214/aos/1176345331.
- Bernardo, Smith (2000): *Bayesian Theory*. Wiley doi:10.1002/9780470316870.
- Jaynes (2003): *Probability Theory: The Logic of Science*. Cambridge University Press doi:10.1017/CBO9780511790423.
- MacKay (2005): *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press https://www.inference.org.uk/itila/book.html.
- Porta Mana (2025): *What's special about 89% credibility intervals?* doi:10.5281/zenodo.17072199.

See Also

learn(), which generates the learnt objects required by Pr().

plot.probability() to plot probabilities and quantiles calculated by Pr().

hist.probability() to plot histograms of the probability distributions calculated by Pr().

print.probability() to print the main elements of the probabilities calculated by Pr().

qPr() to calculate quantiles for a specific variate, that is, the variate values having given probabilities.

rPr() to generate datapoints.

Examples

```
## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## ## Example 1:
## Calculate the probability that an unknown penguin from this population
## is of species 'Adelie'

probs <- Pr(
  Y = data.frame(species = 'Adelie'),
  learnt = learnt, parallel = 1
)

## display the probability value
probs$values

## the full-population frequency of 'Adelie' penguins is unknown;
## display the 5.5%- and 94.5%-probability values
## for such frequency
probs$quantiles[, , c('5.5%', '94.5%')]

## we can also plot the probability distribution for this full-population frequency
hist(probs, legend = 'topright')


## ## Example 2:
## Calculate the 3 probabilities that an unknown penguin from this population
## is of species 'Adelie', 'Chinstrap', 'Gentoo'

probs <- Pr(
  Y = data.frame(species = c('Adelie', 'Chinstrap', 'Gentoo')),
  learnt = learnt, parallel = 1
)

## display the 3 probability values
probs$values

## the full-population frequencies of the three species are unknown;
## display the 5.5%- and 94.5%-probability values
```

```
## for such frequencies
probs$quantiles[, , c('5.5%', '94.5%')]

## plot the probabilities and quantiles
plot(probs)

## plot the probability distribution for the full-population frequency
## of each species
hist(probs)

## ## Example 3:
## Calculate the probability that an unknown penguin is of species 'Adelie'
## GIVEN that its bill length is 43 mm

probs <- Pr(
  Y = data.frame(species = 'Adelie'),
  X = data.frame(bill_len = 43),
  learnt = learnt, parallel = 1
)

## display the probability value
probs$values

## the full-subpopulation frequency of 'Adelie' penguins,
## among penguins having bill length of 43 mm, is unknown;
## display the 5.5%- and 94.5%-probability values
## for such conditional frequency
probs$quantiles[, , c('5.5%', '94.5%')]


## ## Example 4:
## Calculate the probability that
## an unknown penguin is of species 'Adelie' AND its bill length is 43 mm

probs <- Pr(
  Y = data.frame(species = 'Adelie', bill_len = 43),
  learnt = learnt, parallel = 1
)

## display the probability value
probs$values

## display the 5.5%- and 94.5%-probability values
## for the full-population frequency of 'Adelie' penguins with 43 mm bills
probs$quantiles[, , c('5.5%', '94.5%')]


## ## Example 5:
## Calculate the 3 x 2 probabilities for the 3 species
## GIVEN bill-lengths of 43 mm and 44 mm

Y <- data.frame(species = c('Adelie', 'Chinstrap', 'Gentoo'))
```

```
X <- data.frame(bill_len = c(43, 44))

probs <- Pr(Y = Y, X = X, learnt = learnt, parallel = 1)

## display the 3 x 2 probability values
probs$values

## display the 5.5%- and 94.5%-probability values
## for the full-population joint frequencies
probs$quantiles[, , c('5.5%', '94.5%')]

## plot the probabilities and quantiles
plot(probs)


## ## Example 6:
## Calculate the 3 x 2 joint probabilities for the 3 species
## AND bill-lengths of 43 mm and 44 mm

Y <- expand.grid(
  species = c('Adelie', 'Chinstrap', 'Gentoo'),
  bill_len = c(43, 44)
)

probs <- Pr(Y = Y, learnt = learnt, parallel = 1)

## display the 6 joint-probability values
probs$values

## display the 5.5%- and 94.5%-probability values
## for the full-population joint frequencies
probs$quantiles[, , c('5.5%', '94.5%')]
```

`print.probability` *Print an object of class "probability"*

Description

This `base::print()` method is a utility to display selected elements of a "probability" object obtained with `Pr()`; typically its posterior probabilities (element `$values`) and their variabilities (element `$quantiles`). If the `Y` or `X` variates are joint variates, this method also allow to display only selected values of them

Usage

```
## S3 method for class 'probability'
print(x, elements = NULL, subset = NULL, digits = TRUE, ...)
```

Arguments

x	Object of class "probability", obtained with Pr().
elements	character or integer vector, or NULL (default): elements of the "probability" object to display. The syntax is the same as with [. If NULL, the elements $values and $quantiles are displayed together in a special way.
subset	Named list or named vector: which variate values to display. For the variates corresponding to the names in this list, only the vector of values corresponding to that variate is displayed.
digits	positive number or NULL or TRUE (default): minimal number of significant digits, see base::print.default(). If value is TRUE, then the significant digits for elements $values and $quantiles are determined from their respective $values.MCaccuracy and $quantiles.MCaccuracy elements of the probability object, see Pr(); whereas $samples elements use 2 significant digits.
...	Other parameters to be passed to base::print().

Value

Its x argument, invisibly; see base::print().

See Also

Pr() to calculate posterior probabilities and quantiles.

plot.probability() to plot probabilities and quantiles calculated by 'Pr()'. hist.probability() to plot the variability of the probabilities as a distribution.

Examples

```
## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## Calculate the 3 x 2 probabilities for the 3 species
## given bill-lengths of 43 mm and 44 mm

Y <- data.frame(species = c('Adelie', 'Chinstrap', 'Gentoo'))
X <- data.frame(bill_len = c(43, 44))

probs <- Pr(Y = Y, X = X, learnt = learnt, parallel = 1)

## display the values and variabilities of these probabilities
print(probs)

## diplay 'values' only, and only for the species value 'Gentoo'
print(probs, elements = 'values', subset = list(species = 'Gentoo'))
```

prova.data *Write and read CSV files in* **Prova**

Description

Utility functions to read and write CSV files in the format required by **Prova**

Usage

```
pwrite.csv(x, file, ...)

pread.csv(file, ...)
```

Arguments

x	The object to be written. Preferably a matrix or data frame; if not, it is attempted to coerce x to a data frame. See utils::write.csv().
file	Either a character naming a file or a connection open for writing or reading. See utils::write.csv() and utils::read.csv().
...	Other arguments to be passed to utils::write.csv() or utils::read.csv(). Arguments 'row.names', 'quote', 'na', 'na.strings', 'tryLogical', 'sep', 'dec' are not allowed.

Details

The functions learn() and metadatatemplate() accept CSV files formatted as follows:

- Decimal values should be separated by a *dot*; no comma should be used to separate thousands etc. Example: 86342.75.
- Character and names should be quoted in single or double quotes. Example: "female".
- Values should be separated by *commas*, not by tabs or semicolons.
- Missing values should be simply *empty*, not denoted by "NA", "missing", "-", or similar.
- Preferably there should not be factors (see base::factor); use character names instead.

The utility functions pwrite.csv() and pread.csv() are wrappers to utils::write.csv() and utils::read.csv() that set appropriate default parameters according to the formatting rules above.

Value

pread.csv returns a data frame containing a representation of the data in the file; see utils::read.csv(). pwrite.csv' returns NULL‘ invisibly.

See Also

metadatatemplate() to help writing metadata files.

learn(), which needs a metadata data-frame or CSV file.

Examples

```
## Save the 'penguins' dataset in a (temporary) file
filename <- tempfile(fileext = '.csv')

pwrite.csv(penguins, file = filename)

## check first few lines of the raw file
writeLines(readLines(filename, n = 10))
```

---

qPr *Calculate quantiles*

Description

This function calculates the quantiles of posterior probabilities and posterior conditional probabilities. It also outputs the variability of such quantiles if more training data were available.

Usage

```
qPr(
  p = c(0.25, 0.5, 0.75),
  Yname,
  X = NULL,
  learnt,
  tails = NULL,
  priorY = NULL,
  nsamples = "all",
  quantiles = c(0.055, 0.5, 0.945),
  parallel = TRUE,
  sep = ",",
  solidus = "|",
  verbose = FALSE,
  keepYX = TRUE,
  tol = .Machine$double.eps * 10
)
```

Arguments

p	Numeric vector of probability levels. Default: `c(0.25, 0.5, 0.75)`.
Yname	Character vector: name of variate whose quantiles will be computed.
X	Matrix or data.table or `NULL` (default): set of values of variates on which we want to condition. If `NULL`, no conditioning is made (except for conditioning on the learning dataset and prior assumptions). One variate per column, one set of values per row.
learnt	Either a character with the name of a directory or full path for a 'learnt.rds' object, produced by the `learn()` function, or such an object itself.

tails	Named vector or list, or NULL (default). The names must match some or all of the variates in arguments X. For variates in this list, the probability conditional is understood in a semi-open interval sense: $X \leq x$ or $X \geq x$, an so on. See analogous argument in Pr().
priorY	Reserved for use in future versions of the package.
nsamples	Integer or NULL or 'all' (default): desired number of samples of the variability of the quantile for Y. If NULL, no samples are reported. If 'all' (or Inf), all samples obtained by the learn() function are used.
quantiles	Numeric vector, between 0 and 1, or NULL: desired quantiles of the variability of the quantile for Y. Default c(0.055, 0.25, 0.75, 0.945), that is, the 5.5%, 25%, 75%, 94.5% quantiles (these are typical quantile values in the Bayesian literature: they give 50% and 89% credibility intervals, which correspond to 1 shannons and 0.5 shannons of uncertainty). If NULL, no quantiles are calculated.
parallel	Logical or positive integer or cluster object. TRUE (default): use roughly half of available cores; FALSE: use serial computation; integer: use this many cores. It can also be a cluster object previously created with parallel::makeCluster(); in this case the parallel computation will use this object.
sep	character, default ',': character to separate variate names and values
solidus	character, default '|': character prepended to names of the variates in the conditional (typically the X variates).
verbose	Logical, default FALSE: give messages about parallel processing?
keepYX	Logical, default TRUE: keep a copy of the Y and X arguments in the output? This is used for the plot method.
tol	numeric positive: tolerance in the calculation of quantiles. Default: .Machine$double.eps * 10 (typically 2.22045e-15).

Details

This function calculates the quantiles of $\Pr(Y = y|X = x, \text{data})$ or of $\Pr(Y = y|X \leq x, \text{data})$ or combinations thereof, at specified cumulative-probability levels. In other words, it calculates the values of Y having specified cumulative probabilities or conditional probabilities. It also calculates the variability of those quantiles if more learning data were provided. It is somewhat analogous to the q-variants of R distribution functions, such as stats::qnorm(). The variability can be expressed in the form of quantiles, samples, or both, as in the Pr() function. If several joint values are given for the probability levels and for X, the function creates a 2D grid of results for all possible combinations of the given probability levels and X values. Each variate in the argument X can be specified either as a point-value $X = x$ or as a left-open interval $X \leq x$ or as a right-open interval $X \geq x$, through the argument tails.

Value

A list of the following elements:

- values: a matrix with the requested Y-quantiles p conditional on the requested X-values in X, for all combinations of p (rows) and X (columns).
- quantiles (possibly NULL): an array with the variability quantiles (3rd dimension of the array) for the quantiles of the value element.

- `samples` (possibly `NULL`): an array with the variability samples (3rd dimension of the array) for such quantiles.
- `Y`, `X`: copies of the `Y` and `X` arguments.

References

- Porta Mana (2025): *What's special about 89% credibility intervals?* doi:10.5281/zenodo.17072199.

See Also

`learn()`, which generates the `learnt` objects required by `qPr()`.

`Pr()` to calculate joint and conditional probabilities.

`rPr()` to generate datapoints.

Examples

```
### WARNING: the following examples, if run, might even take a minute or more.


## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## ## Example 1:
## Calculate the 5.5%-, 50%-, and 94.5%-quantiles for the variate "bill lengt",
## that is, the values of "bill length" having such cumulative probabilities

quants <- qPr(
  Yname = 'bill_len',
  learnt = learnt, parallel = 1
)

## display the quantile values
quants$values

## verify these values using Pr():
probs <- Pr(
  Y = data.frame(bill_len = c(quants$values)),
  tails = list(bill_len = -1),
  learnt = learnt, parallel = 1
)

## the cumulative probabilities are indeed 0.055, 0.5, 0.945 within numerical error:
probs$values

## display the variability about the quantiles
quants$quantiles


## ## Example 2:
```

```
## Calculate the 5.5%-, 50%-, and 94.5%-quantiles for the variate "bill lengt",
## for the subpopulation of species 'Adelie'

quants <- qPr(
  Yname = 'bill_len',
  X = data.frame(species = 'Adelie'),
  learnt = learnt, parallel = 1
)

## display the quantile values
quants$values

## verify these values using Pr():
probs <- Pr(
  Y = data.frame(bill_len = c(quants$values)),
  X = data.frame(species = 'Adelie'),
  tails = list(bill_len = -1),
  learnt = learnt, parallel = 1)

## the cumulative probabilities are indeed 0.055, 0.5, 0.945 within numerical error:
probs$values
```

---

rPr *Generate datapoints*

Description

This function generates datapoints of chosen joint variates, according to posterior probabilities and posterior conditional probabilities.

Usage

```
rPr(
  n,
  Ynames,
  X = NULL,
  learnt,
  tails = NULL,
  mcsamples = NULL,
  parallel = NULL
)
```

Arguments

n	Positive integer: number of samples to draw.
Ynames	Character vector: names of variates to draw jointly

X	List or data.table or NULL: set of values of variates on which we want to condition the joint probability for Y. If NULL (default), no conditioning is made. Any rows beyond the first are discarded
learnt	Either a character with the name of a directory or full path for a 'learnt.rds' object, produced by the learn() function, or such an object itself.
tails	Named vector or list, or NULL (default). The names must match some or all of the variates in arguments X. For variates in this list, the probability conditional is understood in an semi-open interval sense: $X \leq x$ or $X \geq x$, an so on. See analogous argument in Pr().
mcsamples	Vector of integers, or 'all', or NULL (default): which Monte Carlo samples calculated by the learn() function should be used to draw the variate values. The default is to choose a random subset if n is smaller than their number, otherwise to recycle them as necessary.
parallel	Not used: this function does not use parallelization.

Details

This function generates datapoints according to the posterior probability $\Pr(Y = y|X = x, \text{data})$ or $\Pr(Y = y|X \leq x, \text{data})$ or combinations thereof, for the variates specified in the argument Y, and conditional on the variate values specified in the argument X. It is somewhat analogous to the r-variants of R distribution functions, such as stats::rnorm(). If X is omitted or NULL, then the posterior probability $\Pr(Y|\text{data})$ is used. Each variate in the argument X can be specified either as a point-value $X = x$ or as a left-open interval $X \leq x$ or as a right-open interval $X \geq x$, through the argument tails.

Value

A data frame of joint draws of the variates Ynames from the posterior distribution, conditional on X. The row names of the data frame report the Monte Carlo sample (from learn()) used for that draw, and the total number of draws from that sample so far.

See Also

learn(), which generates the learnt objects required by qPr().

Pr() to calculate joint and conditional probabilities.

qPr() to calculate quantiles.

Examples

```
## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## ## Example 1:
## Generate 10 values of the 'species' variate,
## according to the frequency distribution estimated from the data

datapoints <- rPr(
```

```
  n = 10,
  Ynames = 'species',
  learnt = learnt
)

c(datapoints)


## ## Example 2:
## Generate 5 joint values of the 'species' and 'bill_len' variates.

datapoints <- rPr(
  n = 5,
  Ynames = c('species', 'bill_len'),
  learnt = learnt
)

print(datapoints, row.names = FALSE) ## row names give MCMC information


## ## Example 3:
## Generate 5 values of the 'species' variate,
## for the subpopulation of penguins having bill length shorter than 40 mm

datapoints <- rPr(
  n = 5,
  Ynames = 'species',
  X = data.frame(bill_len = 40),
  tails = list(bill_len = -1),
  learnt = learnt
)

c(datapoints)
```

`vrtgrid` *Create a grid of values for a variate*

Description

This function creates a set of values for a variate, based on the information from data and metadata stored in a `learnt` object, created by the `learn()` function. The set of values depends on the type of variate (nominal or continuous, rounded, and so on, see metadata). The range of values is chosen to include, and extend slightly beyond, the range observed in the data used in the `learn()` function. Variate domains are always respected.

Usage

```
vrtgrid(vrt, learnt, length.out = 129)
```

Arguments

vrt	Character: name of the variate, must match one of the names in the metadata file provided to the learn() function.
learnt	Either a character with the name of a directory or full path for a 'learnt.rds' object, produced by the learn() function, or such an object itself.
length.out	Numeric, positive (default 129): number of values to be created; used only for continuous, non-rounded variates (see metadata).

Value

A numeric or character vector of values.

See Also

learn(), which generates the learnt objects required by vrtgrid().

Pr() to calculate probabilities and their variability.

plot.probability() to plot probabilities and quantiles calculated by Pr().

Examples

```
## Load the example `learnt` object calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
learnt <- learntExample

## set of values for the variate "species";
## since this variate is of a nominal kind, all values are included
valuesSpecies <- vrtgrid(vrt = 'species', learnt = learnt)

print(valuesSpecies)

## create a set of values for the variate "bill length";
## this variate is continuous and rounded, only realistic values are included
valuesBill <- vrtgrid(vrt = 'bill_len', learnt = learnt)

range(valuesBill)

## let's take a subset of these values, to speed up computation
valuesBill <- valuesBill[seq(to = length(valuesBill), length.out = 65)]

## calculate the conditional probabilities for the 'bill_len' values above,
## given the values of 'species'
probs <- Pr(
  Y = data.frame(bill_len = valuesBill),
  X = data.frame(species = valuesSpecies),
  learnt = learnt, parallel = 1
)

## plot the conditional probability distributions, and their variability
plot(probs)
```

Index