Package 'prova'

August 21, 2026

Title Nonparametric Probabilistic-Statistical Variate Analysis

Version 2.3.0

Description Calculate posterior joint and conditional probabilities, probability distributions of population frequencies, information-theoretic measures, and expected utilities, by means of Bayesian nonparametrics. Data can be any combination of nominal, ordinal, continuous, censored, rounded types. 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 & 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 3-319-18968-0>. Data-training function requires the package 'Nimble'.

License AGPL (>= 3)

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

Encoding UTF-8

Depends R (>= 4.5.0)

Suggests nimble (>= 1.4.2), knitr, rmarkdown

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

LazyData true

NeedsCompilation no

Author PierGianLuca Porta Mana [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-6070-0784>), Aurora Grefsrud [ctb] (ORCID: <https://orcid.org/0009-0005-9063-4131>), Håkon Mydland [ctb] (ORCID: <https://orcid.org/0009-0004-9558-8894>), Maksim Ohvrill [ctb], Simen Hesthamar Hauge [ctb] (ORCID: <https://orcid.org/0009-0005-6158-2846>)

Maintainer PierGianLuca Porta Mana <pgl@portamana.org>

Repository CRAN

Date/Publication 2026-08-20 23:10:02 UTC

Contents

`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))
```

`exputility` *Calculate expected utilities and their uncertainties*

Description

This functions calculates the expected utilities of each action or decision corresponding to a given utility matrix. The long-run probable utilities are also calculated.

Usage

```
exputility(u, p)
```

Arguments

u	a utility matrix given as a `base::matrix()` or as a `base::data.frame()` (internally converted into a matrix). Each row of the matrix corresponds to a possible action; each row to an uncertain outcome Y. The number of columns must be equal to the number of Y-values of the "prova_pr" (probability) object of argument p.
p	A "prova_pr" (probability) object, obtained from `Pr()`. The number of Y-values of this object must be equal to the number of columens of the utility matrix of argument um.

Details

This function calculates...

Value

A list of the following elements:

- `'value'`: a matrix of the expected utilities of the actions. One row for each action, one column for each value of the conditional X in the probability p.
- `'samples'`: an array of samples of the expeceted utilities that the actions would have, if many more sample data were available. The first dimension corresponds to the actions, the second to the values of the conditional X, and the third the sample index.
- `'value.acc'`: numerical accuracies of `'value'` elements.
- `'optimal'`: list of actions having maximal expected utility, one list element per column of p (that is, its conditional values X). If there are ties, all actions in the tie are reported.
- `'optimal.samples'`: matrix with the probabilities that each action would be chosen as the optimal one, if more data were available. One row for each action, one column for each column of p (that is, its conditional values X).
- `'optimal.samples'`: matrix of samples of actions having maximal expected utility, if many more sample data were available. Each row correspond to a column of p (that is, its conditional values X); each column is a sample. In case of ties, one action is unsystematically selected via `base::sample()`.
- `'X'`, `'tails'`, `'K'`: copies of the homonymous values from the probability object p.

References

- Raiffa (1970): *Decision Analysis: Introductory Lectures on Choices under Uncertainty*, Addison-Wesley https://archive.org/details/decisionanalysis00raif.
- North (1968): *A Tutorial Introduction to Decision Theory* doi:10.1109/TSSC.1968.300114.
- Lindley (1988): *Making Decisions*, Wiley https://www.wiley.com/Making+Decisions%2C+2nd+Edition-p-x000008175.
- Fenton, Neil (2019): *Risk Assessment and Decision Analysis with Bayesian Networks*, CRC doi:10.1201/b21982
- Sox, Higgins, Owens, Schmidler (2024): *Medical Decision Making*, Wiley doi:10.1002/9781119627876.
- Lusted (1968): *Introduction to Medical Decision Making*, Thomas (Springfield, USA).

See Also

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

Examples

```
## Use the example "prova_K" (knowledge) object 'Kexample'
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

## define a utility matrix with four actions,
## and outcomes depending on the variate 'species'
umatrix <- matrix(c(
 1.80, 0.42, 1.60, -0.12, -1.10, 0.20, -0.51, 0.35, -0.49, 0.35, -0.48, 0.62
 ), nrow = 4, ncol = 3, dimnames = list(actions = paste0('A', 1:4), NULL))

print(umatrix)

## Calculate the probability of the 'species outcomes
probs <- Pr(data.frame(species = c('Adelie', 'Chinstrap', 'Gentoo')),
  Kexample)

## Calculate the expected utilities of the actions
eu <- exputility(umatrix, probs)

eu$value

## optimal action:
eu$optimal

## Probabilities for the actions to be judged as optimal
## if many more sample data were available
eu$optimal.probs
```

`hist.prova_mi` *Plot the revisability of an object of class "prova_mi" (mutual information) as a histogram*

Description

The mutual information calculated with the `mutualinfo()` function, and outputted as a "prova_mi" (mutual information) object, has an associated "revisability" that comes from the finite size of the data sample. A much larger sample might reveal a different value of mutual information.

The `hist()` method for a "prova_mi" (mutual information) object is a utility to visualize this kind of revisability, in the form of a distribution: it shows how the mutual information could change, if we collected a much larger (infinite) data sample, and how likely such change would be. The distribution is represented by a histogram formed from samples of revised mutual information. The bin size is chosen according to the Monte Carlo accuracy.

Usage

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

Arguments

x	Object of class "prova_mi" (mutual information), obtained with mutualinfo().
breaks	as in function graphics::hist(), or NULL (default). Value NULL determines the bin width from the Monte Carlo accuracy (roughly speaking, each bin spans two standard deviations).
lty, lwd, col, alpha.f, xlab, ylab, xlim, ylim, main, grid, axes, add	see analogous arguments in graphics::matplot()
alpha.f.fill	Numeric, default 0.125: opacity of the histogram filling. 0 means no filling.
showvalue	Logical, default TRUE: show the mutual information obtained from the current data sample?
...	Other parameters to be passed to pplot().

Value

Invisibly, an object of class "histogram".

See Also

mutualinfo() to calculate mutual information and its revisability.

print.prova_mi()] to plot mutual information and quantiles calculated by mutualinfo()

pplot() (on which hist.prova_mi() is based) for more general plots.

Examples

```
## Use the "prova_K" (knowledge) object 'Kexample',
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

## calculate the mutual information and its revisability
MI <- mutualinfo('species', 'bill_len', Kexample, nv = 2)

## show the possible revisability of the mutual information,
## if a much larger data sample were collected
hist(MI)
```

`hist.prova_pr` *Plot the revisability of an object of class "prova_pr" (probability) as a histogram*

Description

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

- How the probabilities could change, if we collected a much larger (infinite) data sample, 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 "prova_pr" (probability) object is a utility to visualize this kind of revisability, in the form of a distribution. This distribution is represented by a histogram formed from samples of revised proobabilities (or long-run frequencies). The bin size is chosen according to the Monte Carlo accuracy.

Usage

```
## S3 method for class 'prova_pr'
hist(
  x,
  subset = NULL,
  breaks = NULL,
  legend = "topright",
  lty = c(1, 2, 4, 3, 6, 5),
  lwd = 2,
  col = palette(),
  alpha.f = 1,
  alpha.f.fill = 0.125,
  showmean = TRUE,
```

```
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = c(0, NA),
  main = NULL,
  grid = TRUE,
  axes = FALSE,
  add = FALSE,
  ...
)
```

Arguments

x	Object of class "prova_pr" (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	as in function graphics::hist(), or NULL (default). Value NULL determines bin width from the Monte Carlo accuracy (roughly speaking, each bin spans two standard deviations).
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, axes, add	see analogous arguments in graphics::matplot()
alpha.f.fill	Numeric, default 0.125: opacity of the histogram filling, 0 being completely invisible and 1 completely opaque.
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 pplot().

Value

Invisibly, an object of class "histogram".

See Also

Pr() to calculate posterior probabilities and quantiles.

plot.prova_pr() to plot the posterior probabilities.

pplot() (on which hist.prova_pr() is based) for more general plots.

Examples

```
## Use the "prova_K" (knowledge) object 'Kexample',
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'
```

```
## calculate the probability, and its revisability,
## for the value 'Adelie' of the "species" variate
probs <- Pr(data.frame(species = 'Adelie'), Kexample)
probs$value

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

`Kexample` *Example* K*nowledge object produced by learn()*

Description

An example "prova_K" (knowledge) 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 `Kexample` 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 `Kexample`. It is only meant to be used for illustration purposes of the package's capabilities.

Usage

```
Kexample
```

Format

`Kexample`**:**

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.

---

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,
  valueisK = TRUE,
  subsampledata = NULL,
  prior = missing(data) || is.null(data),
  startupMCiterations = 3600,
  minMCiterations = 0,
  maxMCiterations = +Inf,
  maxhours = +Inf,
  ncheckpoints = 12,
  maxrelMCSE = +Inf,
  minESS = 450,
  initES = 8,
  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`	An *additional*, 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 is used to help estimate some hyperparameters. Note that the `auxdata` and `data` datasets should *not* have datapoints in common.
`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 $\mathrm{nsamples} = \mathrm{nchains} \times \mathrm{nsamplesperchain}$.
`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 $\mathrm{nsamples} = \mathrm{nchains} \times \mathrm{nsamplesperchain}$.
`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 $\mathrm{nsamples} = \mathrm{nchains} \times \mathrm{nsamplesperchain}$.
`parallel`	One of the following values: • A "cluster" object previously created with `parallel::makeCluster()`. • Positive integer: create a parallel cluster with this number of nodes (it will be stopped at the end). • `FALSE`: do not use clusters (one node is still generated, in order to eliminate temporary objects from the computation). • `TRUE` (default): use the cluster that was set as default with `parallel::setDefaultCluster()`; if no such object exists, then generate a cluster with as many nodes as in the option "cl.cores"; if this option is unset, then use 2 nodes.
`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'.

`valueisK` Logical or `NULL`: should the `VALUE` returned be the Knowledge 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. 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 8: 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.out'` 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

To use this function, the package **Nimble** needs to be installed.

This function takes as main inputs a set of data and metadata, and computes the full joint probability distribution for new data, including its "revisability". 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 "prova_K" (knowledge) object, typically saved in a `K.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 # let's say 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.out 2>&1 &
```

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

Value

A "prova_K" (knowledge) object, or name of directory containing such an object and other output files, or NULL, depending on argument valueisK.

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 corresponding data are in the file MCtraces.rds.
- plotsamples_learn.pdf, plotquantiles_learn.pdf: show the marginal posterior distributions of each individual variate, together with their "revisability" (as samples or quantiles).
- log-1.out, log-2.out, ... 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.
- Fortini, Petrone (2024): *Prediction-based uncertainty quantification for exchangeable sequences*, doi:10.1098/rsta.2022.0142.
- 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-031219-0
- 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:
K <- 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 `K` object:
str(K)
```

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

Description

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.**

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`.

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`. The `domainmin` field is also used for a *left-censored* or *interval-censored* variate, together with the `minincluded` field set to `true`.

`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`. The `domainmax` field is also used for a *right-censored* or *interval-censored* variate, together with the `maxincluded` field set to `true`.

`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 *censored* variate has values larger than a given amount all grouped together, and similarly for values smaller than a given amount; for example, the "age" variate in a dataset might group all ages under 18 into a value "18 or less", and all those above 67 into "67 or more". In this case, "18" will go into the `domainmin` field, and `minincluded` must be set to `true`; likewise "67" will go into the `domainmax` field, and `maxincluded` must be set to `true`. (b) 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`). 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)
```

`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.

`meta_penguins` *Metadata file for "penguins" dataset*

Description

A data frame containing the prior information about all variates of the penguins dataset.

Usage

```
meta_penguins
```

Format

`metadataExample`:
A data frame with 8 rows and 10 columns.

Value

No return value.

See Also

datasets::penguins dataset.

`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.

Examples

```
print(meta_penguins)
```

mutualinfo *Calculate mutual information between groups of joint variates*

Description

Functions for calculating the mutual information between two grops of joint variates, as well as its revisability. Function `mutualinfo()` can be used for any variates, but is slower and potentially less accurate. Function `mutualinfoF()` is meant to be used with variates having a finite domain, but is extremely faster and more accurate. See "Details"

Usage

```
mutualinfo(
  Y1names,
  Y2names,
  X = NULL,
  K = NULL,
  tails = NULL,
  quantiles = c(0.055, 0.25, 0.75, 0.945),
  ns = NULL,
  nv = 12,
  unit = "Sh",
  parallel = TRUE,
  sep = ",",
  solidus = "|",
  verbose = FALSE,
  keepX = TRUE
)

mutualinfoF(
  Y1names,
  Y2names,
  X = NULL,
  K = NULL,
  tails = NULL,
  quantiles = c(0.055, 0.25, 0.75, 0.945),
  unit = "Sh",
  parallel = TRUE,
  sep = ",",
  solidus = "|",
  verbose = FALSE,
  keepX = TRUE
)
```

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.
`K`	A "prova_K" (knowledge) object produced by `learn()`. It can also be a path to a 'K.rds' file containing such object, or to a directory containing one.
`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()`.
`quantiles`	Numeric vector, between 0 and 1: desired quantiles of the revisability of the mutual information. Default `c(0.055, 0.25, 0.75, 0.945)`, that is, the 5.5%, 25%, 75%, 94.5% quantiles. See similar argument in `Pr()`.
`ns`	Integer or `Inf` or `NULL` (default): number of Monte Carlo samples in the "prova_K" (knowledge) object to use for calculating the mutual information. If `Inf` or `NULL`, use all Monte Carlo samples available in the "prova_K" (knowledge) object.
`nv`	Integer, default 12: number of *duplicates* of Monte Carlo samples in the "prova_K" (knowledge) object to use for calculating the revisability of the mutual information.
`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`	One of the following values: • A "cluster" object previously created with `parallel::makeCluster()`. • Positive integer: create a parallel cluster with this number of nodes (it will be stopped at the end). • `FALSE`: do not use clusters (one node is still generated, in order to eliminate temporary objects from the computation). • `TRUE` (default): use the cluster that was set as default with `parallel::setDefaultCluster()`; if no such object exists, then generate a cluster with as many nodes as in the option "cl.cores"; if this option is unset, then use 2 nodes.
`sep`	character, default `','`: character to separate the output's variate names and values.
`solidus`	character, default `'|'`: character prepended to the output's names of the variates in the conditional (typically the `X` variates).
`verbose`	Logical, default `FALSE`: give messages about parallel processing?
`keepX`	Logical, default `TRUE`: keep a copy of the `X` argument in the output? This is used for `hist.prova_mi()`.

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 $X = x$ and the knowledge K about data and metadata, is given by

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

an expression which can also be written in several other equivalent ways. If the variates involved are continuous, the sums are replaced by integrals. Mutual information is a model-free information-theoretic measure of association, that is, it 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 `rGauss` in the "Value" section.

The functions `mutualinfo()` and `mutualinfoF()` calculate the mutual information 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 $\Pr(Y_1|K)$ 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`.

Function `mutualinfo()` computes the quantities above via Monte Carlo integration; that is, the sums or integrals are approximated by averages over samples drawn with appropriate probabilities. The computation can take tens of minutes if not hours; it can be sped up by using more nodes (if available) in parallel, through the argument `parallel =`. This function should be used if Y_1 or Y_2 (arguments `Y1names` and `Y2names`) include *continous* variates (see metadata).

Function `mutualinfoF()` computes the quantities above by calculating all required probabilities (a finite number) and performing the exact sums. This can only be done for variates with finite domains. If continuous variates are involved, a set of probabilities is calculated on a finite grid of their domain; for this reason the results may be grossly in error. This function should be used if Y_1 or Y_2 (arguments `Y1names` and `Y2names`) include *only* variates with finite domains, typically nominal or ordinal variates (see metadata).

Value

An object of class "prova_mi" (mutual information), which is a list consisting of the following elements:

- `'value'`, the mutual information between (joint) variates `Y1names` and (joint) variates `Y2names`.
- `'quantiles'`, a vector with the revisability quantiles for the mutual information.
- `'value.acc'`, `quantiles.acc` number and vector with the numerical accuracies (roughly speaking a standard deviation) of the Monte Carlo calculation for the `'value'` and the `'quantiles'` elements.
- `'samples'`, a vector with the revisability samples for the mutual information.
- `'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' 'X'`, `'tails'`: copies of the homonymous input arguments.
- `'K'`: name of the "prova_K" (knowledge) object used in the calculation.

See Also

`print.prova_mi()`] to plot mutual information and quantiles calculated by `mutualinfo()`

`hist.prova_mi()` to plot the revisability of the mutual information.

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

`learn()`, which generates the "prova_K" (knowledge) objects required by `mutualinfo()`.

Examples

```
## Use the example "prova_K" (knowledge) object 'Kexample'
## calculated from the "penguins" dataset;
## variates: 'species' (nominal, finite domain)
## and 'bill_len' (continuous rounded, infinite domain)

## Mutual information between the two variates;
## use mutualinfo() because 'bill_len' has infinite domain;
## set nv = 2 to reduce accuracy but also computation time
MI <- mutualinfo('species', 'bill_len',  Kexample, nv = 2)

## Print mutual information, its accuracy, and its revisability
print(MI)
```

`plot.prova_eu` *Plot an object of class "prova_eu" (expected utility) and its revisability*

Description

This `base::plot()` method is a utility to plot the expected utilities obtained with `exputility()`, as well as their revisabilities.

Usage

```
## S3 method for class 'prova_eu'
plot(
  x,
  type = "b",
  lty = c(1, 2, 4, 3, 6, 5),
  pch = c(1, 2, 0, 5, 6, 3),
  lwd = 2,
  col = palette(),
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  legend = "topright",
  add = FALSE,
  alpha.f = 1,
  grid = TRUE,
  lwd.grid = NULL,
  col.grid = "#00000022",
  axes = FALSE,
```

```
  main = NULL,
  type.spread = "b",
  lty.spread = 1,
  lwd.spread = 1,
  alpha.f.spread = NULL,
  nsamples.spread = 360,
  ...
)
```

Arguments

x	Object of class "prova_eu" (expected utility), obtained with exputility().
type	Character vector (default 'b') or list indicating the type of plot for the main probability distribution; see base::plot().
lty	Analogous to argument lty (line style) in graphics::matplot(), used for the main probability distributions.
pch, col, xlab, ylab, main, xlim, ylim, grid, axes, add, lwd.grid, col.grid	see analogous arguments in graphics::plot.default() and graphics::matplot().
lwd	Analogous to argument lwd (line width) in graphics::matplot(), used for the main probability distributions.
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 'topright'.
alpha.f	Numeric, default 1: opacity of the colours of lines or markers, 0 being completely invisible and 1 completely opaque.
type.spread	character vector (default 'b') or list indicating the type of plot for the revisability samples; see argument type.
lty.spread	Same as parameter lty (line style), but for the line type of the revisability samples.
lwd.spread	Same as parameter lwd (line width), but for the line type of the revisability samples.
alpha.f.spread	Numeric or NULL (default): opacity of the quantile bands or of the long-run-frequency samples, similar to alpha.f. NULL determines a value dependent on the number of samples (more samples, less opacity).
nsamples.spread	Integer, default 360: number of samples of long-run frequencies to display.
...	Other parameters to be passed to pplot().

Details

The x-axis spans the possible actions, and the y-axis their expected utilities. Their revisabilities are shown as an ensemble of 360 (default number) expected-utility curves; the number of samples in the ensemble is indicated beside the left y-axis. If any conditioning variate X was used for the probabilities, $\Pr(\dots|X = x, \dots)$, then one such plot is displayed for each conditioning value x.

The probability that an action would still be considered optimal, if many moro learning data were collected, is indicated above the x-axis label corresponding to that action. An asterisk * marks the optimal actions. If any conditioning variate X was used, then one such probability is shown for each conditioning value.

Value

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

See Also

exputility() to calculate expected utilities and their revisability.

print.prova_eu() to print a summary of expected utilities and their revisability.

pplot() (on which plot.prova_eu() is based) for more general plots.

Examples

```
## Use the example "prova_K" (knowledge) object 'Kexample'
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

## define a utility matrix with four actions,
## and outcomes depending on the variate 'species'
umatrix <- matrix(c(
 1.80, 0.42, 1.60, -0.12, -1.10, 0.20, -0.51, 0.35, -0.49, 0.35, -0.48, 0.62
 ), nrow = 4, ncol = 3, dimnames = list(actions = paste0('A', 1:4), NULL))

print(umatrix)

## Calculate the probability of the 'species outcomes
probs <- Pr(data.frame(species = c('Adelie', 'Chinstrap', 'Gentoo')),
  Kexample)

## Calculate the expected utilities of the actions
eu <- exputility(umatrix, probs)

## plot the expected utilities and their revisability
plot(eu)
```

---

plot.prova_pr *Plot an object of class "prova_pr" (probability)*

Description

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

Usage

```
## S3 method for class 'prova_pr'
plot(
  x,
  spread = NULL,
  subset = NULL,
  PvsY = NULL,
  type = NULL,
  lty = c(1, 2, 4, 3, 6, 5),
  pch = c(1, 2, 0, 5, 6, 3),
  lwd = 2,
  col = palette(),
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = c(0, NA),
  legend = "topright",
  add = FALSE,
  alpha.f = 1,
  grid = TRUE,
  lwd.grid = NULL,
  col.grid = "#00000022",
  axes = FALSE,
  ylab2 = NULL,
  main = NULL,
  type.spread = NULL,
  lty.spread = 1,
  lwd.spread = NULL,
  alpha.f.spread = NULL,
  quantiles.spread = NULL,
  nsamples.spread = 360,
  ...
)
```

Arguments

x	Object of class "prova_pr" (probability), obtained with Pr().
spread	One of the values 'quantiles', 'samples', 'none' (equivalent to NA or FALSE), or NULL (default), in which case the revisability available in p is used. This argument chooses how to represent the revisability of the probability; see Pr(). If the requested representation is not available in the object x, then a warning is issued and no revisability 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.

`type` NULL (default) or character vector or list indicating the type of plot for the main probability distribution; see `base::plot()`. The default NULL value uses type `'l'` (lines) for continuous variates, and `'b'` (points and lines) for discrete variates.

`lty` Analogous to argument `lty` (line style) in `graphics::matplot()`, used for the main probability distributions.

`pch`, `col`, `xlab`, `ylab`, `main`, `xlim`, `ylim`, `grid`, `axes`, `add`, `lwd.grid`, `col.grid`
see analogous arguments in `graphics::plot.default()` and `graphics::matplot()`.

`lwd` Analogous to argument `lwd` (line width) in `graphics::matplot()`, used for the main probability distributions.

`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 `'topright'`.

`alpha.f` Numeric, default 1: opacity of the colours of lines or markers, `0` being completely invisible and 1 completely opaque.

`ylab2` A title for the y-axis on the right side of the plot, if displayed.

`type.spread` NULL (default) or character vector or list indicating the type of plot for the long-run-frequency samples; see. The default NULL value uses type `'l'` (lines) for continuous variates, and `'b'` (points and lines) for discrete variates.

`lty.spread` Same as parameter `lty` (line style), but for the line type of the long-run-frequency samples.

`lwd.spread` Same as parameter `lwd` (line width), but for the line type of the long-run-frequency samples.

`alpha.f.spread` Numeric or NULL (default): opacity of the quantile bands or of the long-run-frequency samples, similar to `alpha.f`. NULL means `0.25` if `spread = 'quantiles'`; and an appropriate value if `spread = 'samples'`,dependent on the number of samples (more samples, less opacity).

`quantiles.spread`
Numeric vector or NULL (default): revisability quantiles to display. Value NULL uses all quantiles available in the x object, or just extreme quantiles if multiple probability curves are shown.

`nsamples.spread`
Integer, default 360: number of samples of long-run frequencies to display.

`...` Other parameters to be passed to `pplot()`.

Details

For a collection of probabilities $\Pr(Y = y|X = x, K)$ with several values y and x, this plot method with argument PvsY set to TRUE shows the probabilities on the y-axis, while the x-axis spans the y values, the curve thus showing the probability distribution (the area underneath is 1, except for possibly omitted tails). One such curve is displayed for each x value. If the argument PvsY is FALSE, then the x-axis spans the x values instead – thus the displayed curve is *not* a probability distribution

(area underneath is not 1). One such curve is displayed for each y value. Which kind of plot is best depends on whether one needs to visualize how the probabilities depend on variate Y or on the conditioning variate X. The default `PvsY` value `NULL` tries to guess the desider behaviour depending on how many different values y and x are contained in the probability object `x`; the variate with the largest number of values is displayed on the x-axis, so as to clutter as little as possible the plot window with multiple curves.

The revisabilities of the probabilities can be visualized in two different ways, determined by the argument `spread`:

- `spread = 'quantiles'`: shows the revisabilities as quantile bands around the probability curves. Which quantiles are shown depends on the `quantiles.spread` argument.
- `spread = 'samples'`: shows the revisabilities as an ensemble of alternative probability curves, which can also be interpreted as possible "long-run frequencies". The number of samples in the ensemble is determined by the argument `nsamples.spread`.
- `spread = 'none'` or `NA` or `FALSE`: does not show any revisability.
- `spread = NULL` (default): use the quantile plot, if quantiles are available; otherwise the ensemble plot, if samples are available; otherwise nothing.

Information about the revisability, such as quantiles or number of samples displayed, is shown beside the left y-axis. While quantile bands look neat, they do not show important details about revised probabilities (long-run frequencies), such as persistent modes. Such details are better displayed in the ensemble plot. It is recommended to always take a look at both visualizations of revisability.

The label on the left y-axis is by default the text `Pr(`Y`|`X`, K)`, displaying the actual Y and X variates present in the probability object `x`. If the displayed probabilities are densities (this means that some Y variates are continuous and not rounded), then lowercase `p` is used istead of `Pr`.

Continuous variates with bounded domains, such as censored variates, may have singular probability values – concentrated probability mass – at the boundary points. When such singular points are present, their probability scale is shown in the *right* y-axis.

Value

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

See Also

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

`hist.prova_pr()` to plot the revisability of the probabilities as a distribution.

`pplot()` (on which `plot.prova_pr()` is based) for more general plots.

Examples

```
## Use the "prova_K" (knowledge) object 'Kexample',
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

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

```
## calculate the probabilities and quantiles
probs <- Pr(valuesBill, Kexample)

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

---

pplot *Plot numeric or character values*

Description

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

Usage

```
pplot(
  x = NULL,
  y = NULL,
  type = NA,
  lty = c(1, 2, 4, 3, 6, 5),
  lwd = 2,
  lend = par("lend"),
  pch = c(1, 2, 0, 5, 6, 3),
  col = palette(),
  xlab = NA,
  ylab = NA,
  xlim = NULL,
  ylim = NULL,
  add = FALSE,
  xdomain = NULL,
  ydomain = NULL,
  alpha.f = 1,
  xjitter = NA,
  yjitter = NA,
  fill = NA,
  alpha.f.fill = 0.25,
  grid = TRUE,
  lwd.grid = NULL,
  col.grid = "#00000022",
  axes = FALSE,
  cex.main = 1,
  ...
)
```

Arguments

x	Numeric or character or list: vectors of x-coordinates. If an element of x is missing, a numeric vector 1:... is created having as many values as the rows of the corresponding element in y.
y	Numeric or character or list: vectors of y-coordinates. If an element of y is missing, a numeric vector 1:... is created having as many values as the rows of the corresponding element in x.
type	Character vector or list indicating the type of plot for each element of x and y. The types of plot are the same as in base::plot(), in particular 'p' for points, 'l' for lines, 'b' for both points and lines, 'c' for empty points joined by lines, 'o' for overplotted points and lines, "n'for empty plot. Additional special types'hx'', ''qx'', ''hy'', ''qy'' are available for plotting histograms and quantile bands; see "Details".
lty, lwd, pch, lend, col, xlab, ylab, add, axes, cex.main	see analogous arguments in graphics::matplot() and graphics::plot.default(); defaults are different (see "Usage").
xlim, ylim	NULL (default) or a vector of two values. If non-NULL and 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 variates represented in the x- and y-axes, in case the x or y argument is a character vector. Note that the domains apply to all elements in x and y. The ordering of the values is respected. If NULL, then unique(x) or unique(y) is used.
alpha.f	Numeric vector or list, default 1: opacity of the line or contour colours, 0 being completely invisible and 1 completely opaque.
xjitter, yjitter	Vector or list of logicals or NA (default): add base::jitter() to x- or y-values? Useful when plotting discrete variates. If NA, jitter is added if both x and y are of character (or factor) class.
fill	Logical or NA (default). For histogram plots (type = 'hx' or 'hy'), value TRUE means fill the histogram, and do not plot its contour; FAlSE means plot only its contour without filling; NA means plot contour and fill. For quantile plots (type = 'qx' or 'qy'), value TRUE do not plot the bands' contours; FAlSE means plot only the contours without filling; NA plots a contour only when the quantile band has zero area (and would be invisible otherwise).
alpha.f.fill	Numeric vector or list, default 0.25: opacity of the filling colours, 0 being completely invisible and 1 completely opaque.
grid	Logical, default TRUE: plot a light grid?
lwd.grid	Numeric, default 1: width of grid lines.
col.grid	Color of grid lines, default '#00000022'. Can be specified in any of the usual ways, see for instance grDevices::col2rgb().
...	Other parameters to be passed to graphics::matplot().

Details

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

- Either or both x and y arguments can be lists. In this case, the first element of x is plotted against the first element of y, and so on, recycling as necessary. This allows for plots having different numbers of base points. The specifications in arguments like type, lty, col, alpha.f, xjitter, and similar apply to each list element in turn.
- Argument x, or each element in x if it is a list, can be of class base::character. In this case, x-axis labels as given in xdomain are used, or the unique values in x if xdomain is NULL. Similarly for y and ydomain. This feature makes it easier to plot nominal and ordinal non-numeric variates.
- Additional plot types are available: 'hx', 'qx', 'hy', 'qy' (internally they use graphics::polygon()):
  - 'hx' plots shaded histograms. Argument x must be a list of breaks, and y a list of counts or densities, for example produced by by graphics::hist(). If the number of rows of x exceeds that of y by one, then this is automatically recognized as a histogram plot.
  - 'qx' plots shaded bands. The first band extends from the line defined by the first column of y, to the line defined by the *last* column; the second band is similarly delimited by the second and second-last columns of y, and so on (if y has an odd number of columns, the central one defines a line rather than a band). The x-values are the corresponding columns of x, recycled if necessary. This plot type is useful for plotting quantile bands calculated with Pr().
  - 'hy', 'qy' are analogous types, but with the roles of x and y switched.
- A jitter can be added to each plot, via the xjitter and yjitter vectors of switches. When either of these arguments is NA, it is internally assessed whether jitter is necessary. This feature makes it easier to generate scatter plots of nominal, ordinal, or rounded-continuous 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 feature is useful in plotting probabilities and histograms, when we want to specify the lower as 0 but want the upper limit to be the the maximum probability.
- Transparency of lines or markers can be specified through argument alpha.f.
- Some defaults are different from base::plot() and graphics::matplot().

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

hist.prova_pr() to plot the revisability of the probabilities as a distribution.

Examples

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


## Scatter plot of 'bill_len' vs 'species':
pplot(x = penguins[, 'species'], y = penguins[, 'bill_len'])

## Scatter plot of 'bill_len' vs 'body_mass';
## in this case the scatter-plot `type = 'p'` must be specified:
pplot(x = penguins[, 'body_mass'], y = penguins[, 'bill_len'], type = 'p')

## Plot y-values having different numbers of x-values
pplot(x = list(1:5, 6:7), y = list(5:1, 6:7))

## Specify only the minimum plotting range
xgrid <- seq(from = -2, to = 2, length.out = 65)
pplot(x = xgrid, y = dnorm(xgrid), ylim = c(0, NA))

## Draw a shaded histogram
## type 'hx' is automatically recognized
histo <- hist(rnorm(1000), breaks = 'FD', plot = FALSE)
pplot(x = histo$breaks, y = histo$density)
```

Pr *Calculate posterior probabilities*

Description

Calculate posterior probabilities and probability densities, cumulative posterior probabilities, and mixtures thereof. Output the "revisability" 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,
  K = NULL,
  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 whose probabilities are sought. One variate per column, one set of values per row.
X	Matrix or data.table or NULL (default): set of values of variates in the conditional of the probability of Y. If NULL, no conditioning is made (besides the conditioning on knowledge K). One variate per column, one set of values per row. See "Details" for the interpretation of unnamed arguments.
K	A "prova_K" (knowledge) object produced by learn(). It can also be a path to a 'K.rds' file containing such object, or to a directory containing one. See "Details" for the interpretation of unnamed arguments.
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 a 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 'lower' or'left' or -1; a right-open interval $Y \geq y$ is indicated by '>=' or 'upper' 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 revisability of the probability for Y. If NULL or 0, 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 revisability 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	One of the following values: • A "cluster" object previously created with parallel::makeCluster(). • Positive integer: create a parallel cluster with this number of nodes (it will be stopped at the end). • FALSE: do not use clusters (one node is still generated, in order to eliminate temporary objects from the computation). • TRUE (default): use the cluster that was set as default with parallel::setDefaultCluster(); if no such object exists, then generate a cluster with as many nodes as in the option "cl.cores"; if this option is unset, then use 2 nodes.

sep	character, default ', ': character to separate the output's variate names and values.
solidus	character, default '|': character prepended to the output's 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 plot.prova_pr().

Details

This function calculates the posterior probability $\Pr(Y = y|X = x, K)$, where $Y = y$ and $X = x$ are two (non overlapping) sets of joint variate values, inputted as data frame arguments Y and X, and K is the information in the data and metadata. It is somewhat analogous to the dxxx-variants and pxxx-variants of R distribution functions. If X is omitted or NULL, then the posterior probability $\Pr(Y = y|K)$ is calculated.

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

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

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

If Pr() is called with two unnamed arguments, Pr(..., ...), then it is interpreted as Pr(Y = ..., K = ...). If it is called with three unnamed arguments, then it is interpreted as either Pr(Y = ..., X = ..., K = ...) or Pr(Y = ..., K = ..., tails = ...), depending on whether the second argument appears to be a "prova_K" (knowledge) object or not.

This function also outputs the "revisability" of the posterior probabilities above, that is, probabilities such as $\Pr(Y = y|X = x, \text{new data}, K)$ 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 revisability 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 revisability.

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 the data corresponding to rows Y is given, the function will calculate the probability $\Pr(Y = y|X = x, K, \text{prior})$ from $\Pr(X = x|Y = y, K)$ and the prior, by means of Bayes's theorem

$$\Pr(Y = y|X = x, K, \text{prior}) = \frac{\Pr(X = x|Y = y, K) \cdot \Pr(Y = y|\text{prior})}{\sum_{y'} \Pr(X = x|Y = y', K) \cdot \Pr(Y = y'|\text{prior})} \, .$$

Important: any values *not* present in the Y data frame are given *zero* prior probability; in other words, the normalization $\sum_{y'}$ only counts the y values appearing in the data frame Y.

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 "prova_pr" (probability), which is a list consisting of the following elements:

- `'value'`: a matrix with the probabilities $\Pr(Y = y|X = x, K)$, 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 revisability quantiles (3rd dimension of the array) for such probabilities.
- `'samples'` (possibly `NULL`): an array with the revisability samples (3rd dimension of the array) for such probabilities.
- `'value.acc'`, `quantiles.acc`: arrays with the numerical accuracies (roughly speaking a standard deviation) of the Monte Carlo calculations for the `'values'` and `'quantiles'` elements.
- `'density'`: numerical vector as long as number of rows in `Y`, used mainly for `plot.prova_pr()`. It is the order of the probability density the `Y`-values: values with `0` are actual probabilities; values with `1` are one-dimensional probability densities $\mathrm{p}(\dots)\,\mathrm{d}y$; values with `2` are two-dimensional probability densities $\mathrm{p}(\dots)\,\mathrm{d}y_1\,\mathrm{d}y_2$; and so on.
- `'Y'`, `'X'`, `'tails'`: copies of the `Y`, `X`, `tails` arguments.
- `'K'`: name of the "prova_K" (knowledge) object used in the calculation.

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.
- Fortini, Petrone (2024): *Prediction-based uncertainty quantification for exchangeable sequences*, doi:10.1098/rsta.2022.0142.
- 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 Knowledge objects required by `Pr()`.

`plot.prova_pr()` to plot probabilities and quantiles calculated by `Pr()`.

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

`print.prova_pr()` 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

```
## Use the "prova_K" (knowledge) object 'Kexample',
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

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

## more explicitly: Pr(Y = data.frame(species = 'Adelie'), K = Kexample)
probs <- Pr(data.frame(species = 'Adelie'), Kexample)

## display the probability value
probs$value

## 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(data.frame(species = c('Adelie', 'Chinstrap', 'Gentoo')),
  Kexample)

## display the 3 probability values
probs$value

## 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

## more explicitly: Pr(Y = ..., X = ..., K = Kexample)
```

```
probs <- Pr(data.frame(species = 'Adelie'), data.frame(bill_len = 43),
  Kexample)

## display the probability value
probs$value

## 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(data.frame(species = 'Adelie', bill_len = 43), Kexample)

## display the probability value
probs$value

## 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, X, Kexample)

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

## 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, Kexample)

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

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

`print.prova_eu` *Print an object of class "prova_eu" (expected utility)*

Description

This `base::print()` method is a utility to display value and revisability of an "prova_mi" (mutual information) object obtained with `mutualinfo()`.

Usage

```
## S3 method for class 'prova_eu'
print(x, elements = NULL, digits = TRUE, edigits = 2, ...)
```

Arguments

`x`	Object of class "prova_eu" (expected utility), obtained with `exputility()`.
`elements`	character or integer vector, or `NULL` (default): elements of the "expected utility" object to display. The syntax is the same as with `[`. If `NULL`, the elements `'value'`, `'value.acc'`, `'optimal.probs'` are displayed together in a special way.
`digits`	positive integer or `NULL` or `TRUE` (default): minimal number of significant digits, see `base::print.default()`. If value is `TRUE`, then the significant digits for element `'value'` are determined from is respective `'value.acc'` (see `exputility()`), according to the rules of the *Guide to the expression of Uncertainty in Measurement*, keeping as many digits as given in parameter `edigits`.
`edigits`	positive integer, default 2: number of significant digits for element `'value'` and `'quantiles'`, if `digits = TRUE`.
`...`	Other parameters to be passed to `base::print()`.

Value

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

References

- Joint Committee for Guides in Metrology (2008): *Guide to the expression of uncertainty in measurement*, doi:10.59161/JCGM100-2008E, https://www.iso.org/sites/JCGM/GUM-JCGM100.htm.

See Also

`exputility()` to calculate expected utilities and their revisability.

Examples

```
## Use the example "prova_K" (knowledge) object 'Kexample'
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

## define a utility matrix with four actions,
## and outcomes depending on the variate 'species'
umatrix <- matrix(c(
 1.80, 0.42, 1.60, -0.12, -1.10, 0.20, -0.51, 0.35, -0.49, 0.35, -0.48, 0.62
 ), nrow = 4, ncol = 3, dimnames = list(actions = paste0('A', 1:4), NULL))

print(umatrix)

## Calculate the probability of the 'species outcomes
probs <- Pr(data.frame(species = c('Adelie', 'Chinstrap', 'Gentoo')),
  Kexample)

## Calculate the expected utilities of the actions
eu <- exputility(umatrix, probs)

## Print the expected utility of each action, its numerical accuracy,
## and the probability that it would be optimal if more data were available

print(eu)
```

`print.prova_K` *Print summary of a "prova_K" (knowledge) object*

Description

This `base::print()` method is a utility to display a summary of a "prova_K" (knowledge) object outputted by `learn()`, internally using `utils::str()`. It also display a summary if `learn()`'s value is only the path to the directory of the rds file containing the "prova_K" (knowledge) object itself (see argument `valueisK =` in `learn()`), by internally retrieving the object. If you want to have a summary of a "prova_K" (knowledge) object in a given directory or rds file, you can explicitly call `print.prova_K(<file path>)`.

Usage

```
## S3 method for class 'prova_K'
print(x, ...)
```

Arguments

x	Object of class "prova_K" (knowledge), output of learn().
...	Other parameters to be passed to utils::str().

Value

Its x argument, invisibly; the structure of the corresponding "prova_K" (knowledge) object, if it exists, is also displayed.

See Also

learn(), which generates a "prova_K" (knowledge) object.

Kexample an example "prova_K" (knowledge) object included with **Prova**.

Examples

```
## Display a summary of the example "prova_K" (knowledge) object
## calculated from the "penguins" dataset
print(Kexample)
```

---

print.prova_mi *Print an object of class "prova_mi" (mutual information) (mutual information)*

Description

This base::print() method is a utility to display value and revisability of an "prova_mi" (mutual information) object obtained with mutualinfo().

Usage

```
## S3 method for class 'prova_mi'
print(x, unit = NULL, elements = NULL, digits = TRUE, edigits = 2, ...)
```

Arguments

x	Object of class "prova_mi" (mutual information), obtained with mutualinfo().
unit	Either NULL, or 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; see analogous argument in mutualinfo(). If NULL (default), the same unit as in the object x is used. Unit conversion is internally performed if this unit is different from that of the object x.

elements	character or integer vector, or NULL (default): elements of the "mutual information" object to display. The syntax is the same as with [. If NULL, the elements 'value', 'value.acc', 'quantiles' are displayed together in a special way.
digits	positive integer or NULL or TRUE (default): minimal number of significant digits, see base::print.default(). If value is TRUE, then the significant digits for element 'value' are determined from is respective 'value.acc' (see mutualinfo()), according to the rules of the *Guide to the expression of Uncertainty in Measurement*, keeping as many digits as given in parameter edigits; whereas 'quantiles' elements uses edigits significant digits.
edigits	positive integer, default 2: number of significant digits for element 'value' and 'quantiles', if digits = TRUE.
...	Other parameters to be passed to base::print().

Value

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

References

- Joint Committee for Guides in Metrology (2008): *Guide to the expression of uncertainty in measurement*, doi:10.59161/JCGM100-2008E, https://www.iso.org/sites/JCGM/GUM-JCGM100.htm.

See Also

mutualinfo() to calculate mutual information.

hist.prova_mi() to plot the revisability of the mutual information.

Examples

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

## Use the "prova_K" (knowledge) object 'Kexample',
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

## Calculate the mutual information between variates 'species' and 'bill_len'
MI <- mutualinfo('species', 'bill_len', Kexample)

## display the value and revisability of the mutual information
print(MI)

## convert to hartleys (base-10 logarithms):
print(MI, unit = 'Hart')
```

print.prova_pr *Print an object of class "prova_pr" (probability)*

Description

This base::print() method is a utility to display selected elements of a "prova_pr" (probability) object obtained with Pr(); typically its posterior probabilities (element 'value') and their revisabilities (element 'quantiles'). If the Y or X variates are joint variates, this method also allow to display only selected values of them. Singular probabilities, such as the probability of a censored value for a continuous variate, are indicated with an asterisk *.

Usage

```
## S3 method for class 'prova_pr'
print(x, elements = NULL, subset = NULL, digits = TRUE, edigits = 2, ...)
```

Arguments

x	Object of class "prova_pr" (probability), obtained with Pr().
elements	character or integer vector, or NULL (default): elements of the "prova_pr" (probability) object to display. The syntax is the same as with [. If NULL, the elements 'value' 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 integer or NULL or TRUE (default): minimal number of significant digits, see base::print.default(). If value is TRUE, then the significant digits for elements 'value' and 'quantiles' are determined from their respective 'value.acc' and 'quantiles.acc' elements of the "prova_pr" (probability) object (see Pr()), according to the rules of the *Guide to the expression of Uncertainty in Measurement*, keeping as many digits as given in parameter edigits; whereas 'samples' elements uses edigits significant digits.
edigits	positive integer, default 2: number of significant digits for elements 'value.acc' and 'quantiles.acc', if digits = TRUE.
...	Other parameters to be passed to base::print().

Value

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

References

- Joint Committee for Guides in Metrology (2008): *Guide to the expression of uncertainty in measurement*, doi:10.59161/JCGM100-2008E, https://www.iso.org/sites/JCGM/GUM-JCGM100.htm.

See Also

Pr() to calculate posterior probabilities and quantiles.

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

Examples

```
## Use the "prova_K" (knowledge) object 'Kexample',
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

## 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, X, Kexample)

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

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

---

qPr *Calculate quantiles*

Description

Calculate the quantiles of posterior probabilities and posterior conditional probabilities. Output the revisability of such quantiles if more training data were available.

Usage

```
qPr(
  p,
  Yname,
  X = NULL,
  K = NULL,
  tails = 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.
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. See "Details" for the interpretation of unnamed arguments.
K	A "prova_K" (knowledge) object produced by learn(). It can also be a path to a 'K.rds' file containing such object, or to a directory containing one. See "Details" for the interpretation of unnamed arguments.
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().
nsamples	Integer or NULL or 'all' (default): desired number of samples of the revisability 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 revisability 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	One of the following values: • A "cluster" object previously created with parallel::makeCluster(). • Positive integer: create a parallel cluster with this number of nodes (it will be stopped at the end). • FALSE: do not use clusters (one node is still generated, in order to eliminate temporary objects from the computation). • TRUE (default): use the cluster that was set as default with parallel::setDefaultCluster(); if no such object exists, then generate a cluster with as many nodes as in the option "cl.cores"; if this option is unset, then use 2 nodes.
sep	character, default ',': character to separate the output's variate names and values.
solidus	character, default '|': character prepended to the output's 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 Yname and X arguments in the output? This is used for plot.prova_pr().
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 $\Pr(Y = y|X = x, K)$ or of $\Pr(Y = y|X \leq x, K)$ 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 revisability of those quantiles if more learning data were provided. It is somewhat analogous to the `qxxx`-variants of R distribution functions. The revisability 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`.

If `qPr()` is called with three unnamed arguments, `qPr(..., ..., ...)`, then it is interpreted as `qPr(p = ..., Yname = ..., K = ...)`.

Value

A list of the following elements:

- `'value'`: 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 revisability quantiles (3rd dimension of the array) for the quantiles of the `'value'` element.
- `'samples'` (possibly `NULL`): an array with the revisability samples (3rd dimension of the array) for such quantiles.
- `'Y'`, `'X'` `'tails'`: copies of the `Y`, `X`, `tails` arguments.
- `'K'`: name of the "prova_K" (knowledge) object used in the calculation.

References

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

See Also

`learn()`, which generates the "prova_K" (knowledge) 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.


## Use the example "prova_K" (knowledge) object 'Kexample'
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

## ## Example 1:
```

```
## Calculate the 25%-, 50%-, and 75%-quantiles for the variate "bill length",
## that is, the values of "bill length" having such cumulative probabilities:

quants <- qPr(c(0.25, 0.5, 0.75), 'bill_len', Kexample)

## display the quantile values
quants$value

## verify these values, within numerical error, using Pr():
probs <- Pr(data.frame(bill_len = c(quants$value)), Kexample,
  tails = list(bill_len = -1))
probs$value

## display the revisability about the quantiles
quants$quantiles


## ## Example 2:
## Calculate the 25%-, 50%-, and 75%-quantiles for the variate "bill length",
## for the subpopulation of species 'Adelie':
quants <- qPr(c(0.25, 0.5, 0.75), 'bill_len', data.frame(species = 'Adelie'),
  Kexample)

## display the quantile values
quants$value

## verify these values, within numerical error, using Pr():
probs <- Pr(data.frame(bill_len = c(quants$value)),
  data.frame(species = 'Adelie'), Kexample, tails = list(bill_len = -1))
probs$value
```

---

rPr *Generate datapoints*

Description

Generates datapoints of chosen joint variates, according to posterior probabilities and posterior conditional probabilities.

Usage

```
rPr(
  n,
  Ynames,
  X = NULL,
  K = NULL,
  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
K	A "prova_K" (knowledge) object produced by learn(). It can also be a path to a 'K.rds' file containing such object, or to a directory containing one.
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().
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 $\Pr(Y = y|X = x, K)$ or $\Pr(Y = y|X \leq x, K)$ 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 rxxx-variants of R distribution functions. If X is omitted or NULL, then the posterior probability $\Pr(Y|K)$ 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.

If rPr() is called with three unnamed arguments, rPr(..., ..., ...), then it is interpreted as rPr(n = ..., Ynames = ..., K = ...).

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 "prova_K" (knowledge) objects required by qPr().

Pr() to calculate joint and conditional probabilities.

qPr() to calculate quantiles.

Examples

```
## Use the example "prova_K" (knowledge) object 'Kexample'
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

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

datapoints <- rPr(10, 'species', Kexample)

c(datapoints)


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

datapoints <- rPr(5, c('species', 'bill_len'), Kexample)

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(5, 'species', data.frame(bill_len = 40), Kexample,
  tails = list(bill_len = 'lower'))

c(datapoints)
```

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

Description

Create a data frame of values for one variate, or a combination of values for several variates.

Usage

```
vrtgrid(vrt, K, length.out = NA)
```

Arguments

`vrt`	Character vector: names of the variates; they must match variate names in the `metadata` file provided to the `learn()` function.
`K`	A "prova_K" (knowledge) object produced by `learn()`. It can also be a path to a 'K.rds' file containing such object, or to a directory containing one.

`length.out` Vector or list of positive integers or NA values, possibly named: number of values to be created for each variate. Elements with names are used for the homonymous variates in `vrt`. Unnamed elements are used for the remaining variates, recycled as necessary. See "Details" for the meaning of NA values. Default NA.

Details

The value ranges are based on the information from data and metadata stored in the Knowledge object (see `learn()`) provided in the `K =` argument; they include, and extend slightly beyond, the ranges observed in the data used in the `learn()` function. Variate domains are always respected.

The set of chosen values, for each variate, depends on the type of variate (nominal or continuous, rounded, and so on, see metadata):

- For a discrete (nominal or ordinal) variate, all possible values are chosen.
- For a continuous, *non-rounded* variate, a number of values as specified in the `length.out` argument; or 257 values if `length.out` is missing or NA.
- For a continuous, *rounded* variate, a number of values as specified in the `length.out` argument; or, if `length.out` is missing or NA, the output values are separated by the variates's rounding interval (field `datastep` in the `metadata`).

The output is a data frame that can be used directly in functions like `Pr()`.

Value

A data frame with columns corresponding to the `vrt` argument, and one row for each combination of the variate values.

See Also

`learn()`, which generates the "prova_K" (knowledge) objects required by `vrtgrid()`.

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

`base::expand.grid()` to create a data frame with combination of specified values of several variates.

`plot.prova_pr()` to plot probabilities and quantiles calculated by `Pr()`.

Examples

```
## Use the "prova_K" (knowledge) object 'Kexample',
## calculated from the "penguins" dataset;
## variates: 'species' and 'bill_len'

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

print(valuesSpecies)

## create a small set of values for the variate "bill length";
## this variate is continuous and rounded
```

```
valuesBill <- vrtgrid('bill_len', Kexample, length.out = 4)

print(valuesBill)

## calculate the conditional probabilities for the 'bill_len' values above,
## given the values of 'species'
probs <- Pr(valuesBill, valuesSpecies, Kexample)


## Create a data frame with all possible combinations of the values above;
## the 'length.out' argument does not apply to the discrete variate 'species'
valuesAll <- vrtgrid(c('species', 'bill_len'), Kexample, length.out = 4)

print(valuesAll)

## base::expand.grid() would give a similar result
valuesAll2 <- expand.grid(
  species = unlist(valuesSpecies), bill_len = unlist(valuesBill)
)

print(valuesAll2)
```

Index