Package 'entropy'

July 22, 2025

Version 1.3.2

Date 2025-04-08

Title Estimation of Entropy, Mutual Information and Related Quantities

Depends R (>= 3.4.0)

Description Implements various estimators of entropy for discrete random variables, including the shrinkage estimator by Hausser and Strimmer (2009), the maximum likelihood and the Millow-Madow estimator, various Bayesian estimators, and the Chao-Shen estimator. It also offers an R interface to the NSB estimator. Furthermore, the package provides functions for estimating the Kullback-Leibler divergence, the chi-squared divergence, mutual information, and the chi-squared divergence of independence. It also computes the G statistic and the chi-squared statistic and corresponding p-values. Furthermore, there are functions for discretizing continuous random variables.

License GPL (>= 3)

URL https://strimmerlab.github.io/software/entropy/

NeedsCompilation no

Author Jean Hausser [aut],
Korbinian Strimmer [aut, cre]

Maintainer Korbinian Strimmer <strimmerlab@gmail.com>

Repository CRAN

Date/Publication 2025-04-07 23:50:02 UTC

Contents

entropy-package 2
discretize 3
entropy 5
entropy.ChaoShen 6
entropy.Dirichlet 8
entropy.empirical 11
entropy.MillerMadow 13
entropy.NSB 14

entropy.plugin 16
entropy.shrink 17
Gstat 19
KL.plugin 21
mi.plugin 23

Index **25**

`entropy-package` *The entropy Package*

Description

This package implements various estimators of the Shannon entropy. Most estimators in this package can be applied in "small n, large p" situations, i.e. when there are many more bins than counts.

The main function of this package is `entropy`, which provides a unified interface to various entropy estimators. Other functions included in this package are estimators of Kullback-Leibler divergence (`KL.plugin`), mutual information (`mi.plugin`) and of the chi-squared divergence (`chi2.plugin`). Furthermore, there are functions to compute the G statistic (`Gstat`) and the chi-squared statistic (`chi2stat`).

If you use this package please cite: Jean Hausser and Korbinian Strimmer. 2009. Entropy inference and the James-Stein estimator, with application to nonlinear gene association networks. J. Mach. Learn. Res. **10**: 1469-1484. Available online from https://jmlr.csail.mit.edu/papers/v10/hausser09a.html.

This paper contains a detailed statistical comparison of the estimators available in this package. It also describes the shrinkage entropy estimator `entropy.shrink`.

Author(s)

Jean Hausser and Korbinian Strimmer (https://strimmerlab.github.io/)

References

See website: https://strimmerlab.github.io/software/entropy/

See Also

`entropy`

discretize *Discretize Continuous Random Variables*

Description

`discretize` puts observations from a continuous random variable into bins and returns the corresponding vector of counts.

`discretize2d` puts observations from a pair of continuous random variables into bins and returns the corresponding table of counts.

Usage

```
discretize( x, numBins, r=range(x) )
discretize2d( x1, x2, numBins1, numBins2, r1=range(x1), r2=range(x2) )
```

Arguments

`x`	vector of observations.
`x1`	vector of observations for the first random variable.
`x2`	vector of observations for the second random variable.
`numBins`	number of bins.
`numBins1`	number of bins for the first random variable.
`numBins2`	number of bins for the second random variable.
`r`	range of the random variable (default: observed range).
`r1`	range of the first random variable (default: observed range).
`r2`	range of the second random variable (default: observed range).

Details

The bins for a random variable all have the same width. It is determined by the length of the range divided by the number of bins.

Value

`discretize` returns a vector containing the counts for each bin.

`discretize2d` returns a matrix containing the counts for each bin.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

See Also

`entropy`.

Examples

```
# load entropy library
library("entropy")

### 1D example ####

# sample from continuous uniform distribution
x1 = runif(10000)
hist(x1, xlim=c(0,1), freq=FALSE)

# discretize into 10 categories
y1 = discretize(x1, numBins=10, r=c(0,1))
y1

# compute entropy from counts
entropy(y1) # empirical estimate near theoretical maximum
log(10) # theoretical value for discrete uniform distribution with 10 bins

# sample from a non-uniform distribution
x2 = rbeta(10000, 750, 250)
hist(x2, xlim=c(0,1), freq=FALSE)

# discretize into 10 categories and estimate entropy
y2 = discretize(x2, numBins=10, r=c(0,1))
y2
entropy(y2) # almost zero

### 2D example ####

# two independent random variables
x1 = runif(10000)
x2 = runif(10000)

y2d = discretize2d(x1, x2, numBins1=10, numBins2=10)
sum(y2d)

# joint entropy
H12 = entropy(y2d )
H12
log(100) # theoretical maximum for 10x10 table

# mutual information
mi.empirical(y2d) # approximately zero


# another way to compute mutual information

# compute marginal entropies
H1 = entropy(rowSums(y2d))
H2 = entropy(colSums(y2d))

H1+H2-H12 # mutual entropy
```

entropy *Estimating Entropy From Observed Counts*

Description

entropy estimates the Shannon entropy H of the random variable Y from the corresponding observed counts y.

freqs estimates bin frequencies from the counts y.

Usage

```
entropy(y, lambda.freqs, method=c("ML", "MM", "Jeffreys", "Laplace", "SG",
   "minimax", "CS", "NSB", "shrink"), unit=c("log", "log2", "log10"), verbose=TRUE, ...)
freqs(y, lambda.freqs, method=c("ML", "MM", "Jeffreys", "Laplace", "SG",
    "minimax", "CS", "NSB", "shrink"), verbose=TRUE)
```

Arguments

y	vector of counts.
method	the method employed to estimate entropy (see Details).
unit	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set unit="log2".
lambda.freqs	shrinkage intensity (for "shrink" option).
verbose	verbose option (for "shrink" option).
...	option passed on to entropy.NSB.

Details

The entropy function allows to estimate entropy from observed counts by a variety of methods:

method="ML": maximum likelihood, see entropy.empirical

method="MM": bias-corrected maximum likelihood, see entropy.MillerMadow

method="Jeffreys": entropy.Dirichlet with a=1/2

method="Laplace": entropy.Dirichlet with a=1

method="SG": entropy.Dirichlet with a=a=1/length(y)

method="minimax": entropy.Dirichlet with a=sqrt(sum(y))/length(y

method="CS": see entropy.ChaoShen

method="NSB": see entropy.NSB

method="shrink": see entropy.shrink

The freqs function estimates the underlying bin frequencies. Note that estimated frequencies are not available for method="MM", method="CS" and method="NSB". In these instances a vector containing NAs is returned.

Value

entropy returns an estimate of the Shannon entropy.

freqs returns a vector with estimated bin frequencies (if available).

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

See Also

entropy-package, discretize.

Examples

```
# load entropy library
library("entropy")

# observed counts for each bin
y = c(4, 2, 3, 0, 2, 4, 0, 0, 2, 1, 1)

entropy(y, method="ML")
entropy(y, method="MM")
entropy(y, method="Jeffreys")
entropy(y, method="Laplace")
entropy(y, method="SG")
entropy(y, method="minimax")
entropy(y, method="CS")
#entropy(y, method="NSB")
entropy(y, method="shrink")
```

---

entropy.ChaoShen *Chao-Shen Entropy Estimator*

Description

entropy.ChaoShen estimates the Shannon entropy H of the random variable Y from the corresponding observed counts y using the method of Chao and Shen (2003).

Usage

```
entropy.ChaoShen(y, unit=c("log", "log2", "log10"))
```

Arguments

y	vector of counts.
unit	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set unit="log2".

Details

The Chao-Shen entropy estimator (2003) is a Horvitz-Thompson (1952) estimator applied to the problem of entropy estimation, with additional coverage correction as proposed by Good (1953).

Note that the Chao-Shen estimator is not a plug-in estimator, hence there are no explicit underlying bin frequencies.

Value

`entropy.ChaoShen` returns an estimate of the Shannon entropy.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

References

Chao, A., and T.-J. Shen. 2003. Nonparametric estimation of Shannon's index of diversity when there are unseen species in sample. Environ. Ecol. Stat. **10**:429-443.

Good, I. J. 1953. The population frequencies of species and the estimation of population parameters. Biometrika **40**:237-264.

Horvitz, D.G., and D. J. Thompson. 1952. A generalization of sampling without replacement from a finite universe. J. Am. Stat. Assoc. **47**:663-685.

See Also

`entropy`, `entropy.shrink`, `entropy.Dirichlet`, `entropy.NSB`.

Examples

```
# load entropy library
library("entropy")

# observed counts for each bin
y = c(4, 2, 3, 0, 2, 4, 0, 0, 2, 1, 1)

# estimate entropy using Chao-Shen method
entropy.ChaoShen(y)

# compare to empirical estimate
entropy.empirical(y)
```

`entropy.Dirichlet` *Dirichlet Prior Bayesian Estimators of Entropy, Mutual Information and Other Related Quantities*

Description

`freqs.Dirichlet` computes the Bayesian estimates of the bin frequencies using the Dirichlet-multinomial pseudocount model.

`entropy.Dirichlet` estimates the Shannon entropy H of the random variable Y from the corresponding observed counts `y` by plug-in of Bayesian estimates of the bin frequencies using the Dirichlet-multinomial pseudocount model.

`KL.Dirichlet` computes a Bayesian estimate of the Kullback-Leibler (KL) divergence from counts `y1` and `y2`.

`chi2.Dirichlet` computes a Bayesian version of the chi-squared divergence from counts `y1` and `y2`.

`mi.Dirichlet` computes a Bayesian estimate of mutual information of two random variables.

`chi2indep.Dirichlet` computes a Bayesian version of the chi-squared divergence of independence from a table of counts `y2d`.

Usage

```
freqs.Dirichlet(y, a)
entropy.Dirichlet(y, a, unit=c("log", "log2", "log10"))
KL.Dirichlet(y1, y2, a1, a2, unit=c("log", "log2", "log10"))
chi2.Dirichlet(y1, y2, a1, a2, unit=c("log", "log2", "log10"))
mi.Dirichlet(y2d, a, unit=c("log", "log2", "log10"))
chi2indep.Dirichlet(y2d, a, unit=c("log", "log2", "log10"))
```

Arguments

`y`	vector of counts.
`y1`	vector of counts.
`y2`	vector of counts.
`y2d`	matrix of counts.
`a`	pseudocount per bin.
`a1`	pseudocount per bin for first random variable.
`a2`	pseudocount per bin for second random variable.
`unit`	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set `unit="log2"`.

Details

The Dirichlet-multinomial pseudocount entropy estimator is a Bayesian plug-in estimator: in the definition of the Shannon entropy the bin probabilities are replaced by the respective Bayesian estimates of the frequencies, using a model with a Dirichlet prior and a multinomial likelihood.

The parameter a is a parameter of the Dirichlet prior, and in effect specifies the pseudocount per bin. Popular choices of a are:

a=0: maximum likelihood estimator (see entropy.empirical)

a=1/2: Jeffreys' prior; Krichevsky-Trovimov (1991) entropy estimator

a=1: Laplace's prior

a=1/length(y): Schurmann-Grassberger (1996) entropy estimator

a=sqrt(sum(y))/length(y): minimax prior

The pseudocount a can also be a vector so that for each bin an individual pseudocount is added.

Value

freqs.Dirichlet returns the Bayesian estimates of the frequencies .

entropy.Dirichlet returns the Bayesian estimate of the Shannon entropy.

KL.Dirichlet returns the Bayesian estimate of the KL divergence.

chi2.Dirichlet returns the Bayesian version of the chi-squared divergence.

mi.Dirichlet returns the Bayesian estimate of the mutual information.

chi2indep.Dirichlet returns the Bayesian version of the chi-squared divergence of independence.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

References

Agresti, A., and D. B. Hitchcock. 2005. Bayesian inference for categorical data analysis. Stat. Methods. Appl. **14**:297–330.

Krichevsky, R. E., and V. K. Trofimov. 1981. The performance of universal encoding. IEEE Trans. Inf. Theory **27**: 199-207.

Schurmann, T., and P. Grassberger. 1996. Entropy estimation of symbol sequences. Chaos **6**:41-427.

See Also

entropy, entropy.shrink, entropy.empirical, entropy.plugin, mi.plugin, KL.plugin, discretize.

Examples

```
# load entropy library
library("entropy")


# a single variable

# observed counts for each bin
y = c(4, 2, 3, 0, 2, 4, 0, 0, 2, 1, 1)

# Dirichlet estimate of frequencies with a=1/2
freqs.Dirichlet(y, a=1/2)

# Dirichlet estimate of entropy with a=0
entropy.Dirichlet(y, a=0)

# identical to empirical estimate
entropy.empirical(y)

# Dirichlet estimate with a=1/2 (Jeffreys' prior)
entropy.Dirichlet(y, a=1/2)

# Dirichlet estimate with a=1 (Laplace prior)
entropy.Dirichlet(y, a=1)

# Dirichlet estimate with a=1/length(y)
entropy.Dirichlet(y, a=1/length(y))

# Dirichlet estimate with a=sqrt(sum(y))/length(y)
entropy.Dirichlet(y, a=sqrt(sum(y))/length(y))


# example with two variables

# observed counts for two random variables
y1 = c(4, 2, 3, 1, 10, 4)
y2 = c(2, 3, 7, 1, 4, 3)

# Bayesian estimate of Kullback-Leibler divergence (a=1/6)
KL.Dirichlet(y1, y2, a1=1/6, a2=1/6)

# half of the corresponding chi-squared divergence
0.5*chi2.Dirichlet(y1, y2, a1=1/6, a2=1/6)


## joint distribution example

# contingency table with counts for two discrete variables
y2d = rbind( c(1,2,3), c(6,5,4) )

# Bayesian estimate of mutual information (a=1/6)
mi.Dirichlet(y2d, a=1/6)
```

```
# half of the Bayesian chi-squared divergence of independence
0.5*chi2indep.Dirichlet(y2d, a=1/6)
```

`entropy.empirical` *Empirical Estimators of Entropy and Mutual Information and Related Quantities*

Description

`freqs.empirical` computes the empirical frequencies from counts `y`.

`entropy.empirical` estimates the Shannon entropy H of the random variable Y from the corresponding observed counts `y` by plug-in of the empirical frequencies.

`KL.empirical` computes the empirical Kullback-Leibler (KL) divergence from counts `y1` and `y2`.

`chi2.empirical` computes the empirical chi-squared divergence from counts `y1` and `y2`.

`mi.empirical` computes the empirical mutual information from a table of counts `y2d`.

`chi2indep.empirical` computes the empirical chi-squared divergence of independence from a table of counts `y2d`.

Usage

```
freqs.empirical(y)
entropy.empirical(y, unit=c("log", "log2", "log10"))
KL.empirical(y1, y2, unit=c("log", "log2", "log10"))
chi2.empirical(y1, y2, unit=c("log", "log2", "log10"))
mi.empirical(y2d, unit=c("log", "log2", "log10"))
chi2indep.empirical(y2d, unit=c("log", "log2", "log10"))
```

Arguments

`y`	vector of counts.
`y1`	vector of counts.
`y2`	vector of counts.
`y2d`	matrix of counts.
`unit`	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set `unit="log2"`.

Details

The empirical entropy estimator is a plug-in estimator: in the definition of the Shannon entropy the bin probabilities are replaced by the respective empirical frequencies.

The empirical entropy estimator is the maximum likelihood estimator. If there are many zero counts and the sample size is small it is very inefficient and also strongly biased.

Value

`freqs.empirical` returns the empirical frequencies.

`entropy.empirical` returns an estimate of the Shannon entropy.

`KL.empirical` returns an estimate of the KL divergence.

`chi2.empirical` returns the empirical chi-squared divergence.

`mi.empirical` returns an estimate of the mutual information.

`chi2indep.empirical` returns the empirical chi-squared divergence of independence.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

See Also

`entropy`, `entropy.plugin`, `KL.plugin`, `chi2.plugin`, `mi.plugin`, `chi2indep.plugin`, `Gstat`, `Gstatindep`, `chi2stat`, `chi2statindep`, `discretize`.

Examples

```
# load entropy library
library("entropy")


## a single variable: entropy

# observed counts for each bin
y = c(4, 2, 3, 0, 2, 4, 0, 0, 2, 1, 1)

# empirical frequencies
freqs.empirical(y)

# empirical estimate of entropy
entropy.empirical(y)


## examples with two variables: KL and chi-squared divergence

# observed counts for first random variables (observed)
y1 = c(4, 2, 3, 1, 6, 4)
n = sum(y1) # 20

# counts for the second random variable (expected)
freqs.expected = c(0.10, 0.15, 0.35, 0.05, 0.20, 0.15)
y2 = n*freqs.expected

# empirical Kullback-Leibler divergence
KL.div = KL.empirical(y1, y2)
KL.div

# empirical chi-squared divergence
```

```
cs.div = chi2.empirical(y1, y2)
cs.div
0.5*cs.div  # approximates KL.div

## note: see also Gstat and chi2stat


## joint distribution of two discrete random variables

# contingency table with counts for two discrete variables
y.mat = matrix(c(4, 5, 1, 2, 4, 4), ncol = 2)  # 3x2 example matrix of counts
n.mat = sum(y.mat) # 20

# empirical estimate of mutual information
mi = mi.empirical(y.mat)
mi

# empirical chi-squared divergence of independence
cs.indep = chi2indep.empirical(y.mat)
cs.indep
0.5*cs.indep # approximates mi

## note: see also Gstatindep and chi2statindep
```

`entropy.MillerMadow` *Miller-Madow Entropy Estimator*

Description

`entropy.MillerMadow` estimates the Shannon entropy H of the random variable Y from the corresponding observed counts `y` using the Miller-Madow correction to the empirical entropy).

Usage

```
entropy.MillerMadow(y, unit=c("log", "log2", "log10"))
```

Arguments

`y`	vector of counts.
`unit`	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set `unit="log2"`.

Details

The Miller-Madow entropy estimator (1955) is the bias-corrected empirical entropy estimate.

Note that the Miller-Madow estimator is not a plug-in estimator, hence there are no explicit underlying bin frequencies.

Value

`entropy.MillerMadow` returns an estimate of the Shannon entropy.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

References

Miller, G. 1955. Note on the bias of information estimates. Info. Theory Psychol. Prob. Methods **II-B**:95-100.

See Also

`entropy.empirical`

Examples

```
# load entropy library
library("entropy")

# observed counts for each bin
y = c(4, 2, 3, 0, 2, 4, 0, 0, 2, 1, 1)

# estimate entropy using Miller-Madow method
entropy.MillerMadow(y)

# compare to empirical estimate
entropy.empirical(y)
```

`entropy.NSB` *R Interface to NSB Entropy Estimator*

Description

`entropy.NSB` estimates the Shannon entropy H of the random variable Y from the corresponding observed counts `y` using the method of Nemenman, Shafee and Bialek (2002).

Note that this function is an R interface to the "nsb-entropy" program. Hence, this needs to be installed separately from https://nsb-entropy.sourceforge.net/.

Usage

```
entropy.NSB(y, unit=c("log", "log2", "log10"), CMD="nsb-entropy")
```

Arguments

y	vector of counts.
unit	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set unit="log2".
CMD	path to the "nsb-entropy" executable.

Details

The NSB estimator is due to Nemenman, Shafee and Bialek (2002). It is a Dirichlet-multinomial entropy estimator, with a hierarchical prior over the Dirichlet pseudocount parameters.

Note that the NSB estimator is not a plug-in estimator, hence there are no explicit underlying bin frequencies.

Value

entropy.NSB returns an estimate of the Shannon entropy.

Author(s)

Jean Hausser.

References

Nemenman, I., F. Shafee, and W. Bialek. 2002. Entropy and inference, revisited. In: Dietterich, T., S. Becker, Z. Gharamani, eds. Advances in Neural Information Processing Systems 14: 471-478. Cambridge (Massachusetts): MIT Press.

See Also

entropy, entropy.shrink, entropy.Dirichlet, entropy.ChaoShen.

Examples

```
# load entropy library
library("entropy")

# observed counts for each bin
y = c(4, 2, 3, 0, 2, 4, 0, 0, 2, 1, 1)

## Not run:
# estimate entropy using the NSB method
entropy.NSB(y) # 2.187774

## End(Not run)

# compare to empirical estimate
entropy.empirical(y)
```

`entropy.plugin` *Plug-In Entropy Estimator*

Description

`entropy.plugin` computes the Shannon entropy H of a discrete random variable with the specified frequencies (probability mass function).

Usage

```
entropy.plugin(freqs, unit=c("log", "log2", "log10"))
```

Arguments

`freqs`	frequencies (probability mass function).
`unit`	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set `unit="log2"`.

Details

The Shannon entropy of a discrete random variable is defined as $H = -\sum_k p(k) \log(p(k))$, where p is its probability mass function.

Value

`entropy.plugin` returns the Shannon entropy.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

See Also

`entropy`, `entropy.empirical`, `entropy.shrink`, `mi.plugin`, `KL.plugin`, `discretize`.

Examples

```
# load entropy library
library("entropy")

# some frequencies
freqs = c(0.2, 0.1, 0.15, 0.05, 0, 0.3, 0.2)

# and corresponding entropy
entropy.plugin(freqs)
```

entropy.shrink *Shrinkage Estimators of Entropy, Mutual Information and Related Quantities*

Description

`freq.shrink` estimates the bin frequencies from the counts `y` using a James-Stein-type shrinkage estimator, where the shrinkage target is the uniform distribution.

`entropy.shrink` estimates the Shannon entropy H of the random variable Y from the corresponding observed counts `y` by plug-in of shrinkage estimate of the bin frequencies.

`KL.shrink` computes a shrinkage estimate of the Kullback-Leibler (KL) divergence from counts `y1` and `y2`.

`chi2.shrink` computes a shrinkage version of the chi-squared divergence from counts `y1` and `y2`.

`mi.shrink` estimates a shrinkage estimate of mutual information of two random variables.

`chi2indep.shrink` computes a shrinkage version of the chi-squared divergence of independence from a table of counts `y2d`.

Usage

```
freqs.shrink(y, lambda.freqs, verbose=TRUE)
entropy.shrink(y, lambda.freqs, unit=c("log", "log2", "log10"), verbose=TRUE)
KL.shrink(y1, y2, lambda.freqs1, lambda.freqs2, unit=c("log", "log2", "log10"),
          verbose=TRUE)
chi2.shrink(y1, y2, lambda.freqs1, lambda.freqs2, unit=c("log", "log2", "log10"),
          verbose=TRUE)
mi.shrink(y2d, lambda.freqs, unit=c("log", "log2", "log10"), verbose=TRUE)
chi2indep.shrink(y2d, lambda.freqs, unit=c("log", "log2", "log10"), verbose=TRUE)
```

Arguments

`y`	vector of counts.
`y1`	vector of counts.
`y2`	vector of counts.
`y2d`	matrix of counts.
`unit`	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set `unit="log2"`.
`lambda.freqs`	shrinkage intensity. If not specified (default) it is estimated in a James-Stein-type fashion.
`lambda.freqs1`	shrinkage intensity for first random variable. If not specified (default) it is estimated in a James-Stein-type fashion.
`lambda.freqs2`	shrinkage intensity for second random variable. If not specified (default) it is estimated in a James-Stein-type fashion.
`verbose`	report shrinkage intensity.

Details

The shrinkage estimator is a James-Stein-type estimator. It is essentially a entropy.Dirichlet estimator, where the pseudocount is estimated from the data.

For details see Hausser and Strimmer (2009).

Value

freqs.shrink returns a shrinkage estimate of the frequencies.

entropy.shrink returns a shrinkage estimate of the Shannon entropy.

KL.shrink returns a shrinkage estimate of the KL divergence.

chi2.shrink returns a shrinkage version of the chi-squared divergence.

mi.shrink returns a shrinkage estimate of the mutual information.

chi2indep.shrink returns a shrinkage version of the chi-squared divergence of independence.

In all instances the estimated shrinkage intensity is attached to the returned value as attribute lambda.freqs.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

References

Hausser, J., and K. Strimmer. 2009. Entropy inference and the James-Stein estimator, with application to nonlinear gene association networks. J. Mach. Learn. Res. **10**: 1469-1484. Available online from https://jmlr.csail.mit.edu/papers/v10/hausser09a.html.

See Also

entropy, entropy.Dirichlet, entropy.plugin, KL.plugin, mi.plugin, discretize.

Examples

```
# load entropy library
library("entropy")

# a single variable

# observed counts for each bin
y = c(4, 2, 3, 0, 2, 4, 0, 0, 2, 1, 1)

# shrinkage estimate of frequencies
freqs.shrink(y)

# shrinkage estimate of entropy
entropy.shrink(y)


# example with two variables
```

```
# observed counts for two random variables
y1 = c(4, 2, 3, 1, 10, 4)
y2 = c(2, 3, 7, 1, 4, 3)

# shrinkage estimate of Kullback-Leibler divergence
KL.shrink(y1, y2)

# half of the shrinkage chi-squared divergence
0.5*chi2.shrink(y1, y2)


## joint distribution example

# contingency table with counts for two discrete variables
y2d = rbind( c(1,2,3), c(6,5,4) )

# shrinkage estimate of mutual information
mi.shrink(y2d)

# half of the shrinkage chi-squared divergence of independence
0.5*chi2indep.shrink(y2d)
```

`Gstat` *G Statistic and Chi-Squared Statistic*

Description

`Gstat` computes the G statistic.

`chi2stat` computes the Pearson chi-squared statistic.

`Gstatindep` computes the G statistic between the empirical observed joint distribution and the product distribution obtained from its marginals.

`chi2statindep` computes the Pearson chi-squared statistic of independence.

Usage

```
Gstat(y, freqs, unit=c("log", "log2", "log10"))
chi2stat(y, freqs, unit=c("log", "log2", "log10"))
Gstatindep(y2d, unit=c("log", "log2", "log10"))
chi2statindep(y2d, unit=c("log", "log2", "log10"))
```

Arguments

`y`	observed vector of counts.
`freqs`	vector of expected frequencies (probability mass function). Alternatively, counts may be provided.

y2d	matrix of counts.
unit	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set unit="log2".

Details

The observed counts in y and y2d are used to determine the total sample size.

The G statistic equals two times the sample size times the KL divergence between empirical observed frequencies and expected frequencies.

The Pearson chi-squared statistic equals sample size times chi-squared divergence between empirical observed frequencies and expected frequencies. It is a quadratic approximation of the G statistic.

The G statistic between the empirical observed joint distribution and the product distribution obtained from its marginals is equal to two times the sample size times mutual information.

The Pearson chi-squared statistic of independence equals the Pearson chi-squared statistic between the empirical observed joint distribution and the product distribution obtained from its marginals. It is a quadratic approximation of the corresponding G statistic.

The G statistic and the Pearson chi-squared statistic are asymptotically chi-squared distributed which allows to compute corresponding p-values.

Value

A list containing the test statistic stat, the degree of freedom df used to calculate the p-value pval.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

See Also

KL.plugin, chi2.plugin, mi.plugin, chi2indep.plugin.

Examples

```
# load entropy library
library("entropy")

## one discrete random variable

# observed counts in each class
y = c(4, 2, 3, 1, 6, 4)
n = sum(y) # 20

# expected frequencies and counts
freqs.expected = c(0.10, 0.15, 0.35, 0.05, 0.20, 0.15)
y.expected = n*freqs.expected


# G statistic (with p-value)
```

```
Gstat(y, freqs.expected) # from expected frequencies
Gstat(y, y.expected) # alternatively from expected counts

# G statistic computed from empirical KL divergence
2*n*KL.empirical(y, y.expected)


## Pearson chi-squared statistic (with p-value)
# this can be viewed an approximation of the G statistic
chi2stat(y, freqs.expected) # from expected frequencies
chi2stat(y, y.expected) # alternatively from expected counts

# computed from empirical chi-squared divergence
n*chi2.empirical(y, y.expected)

# compare with built-in function
chisq.test(y, p = freqs.expected)


## joint distribution of two discrete random variables

# contingency table with counts
y.mat = matrix(c(4, 5, 1, 2, 4, 4), ncol = 2)  # 3x2 example matrix of counts
n.mat = sum(y.mat) # 20


# G statistic between empirical observed joint distribution and product distribution
Gstatindep( y.mat )

# computed from empirical mutual information
2*n.mat*mi.empirical(y.mat)


# Pearson chi-squared statistic of independence
chi2statindep( y.mat )

# computed from empirical chi-square divergence
n.mat*chi2indep.empirical(y.mat)

# compare with built-in function
chisq.test(y.mat)
```

KL.plugin *Plug-In Estimator of the Kullback-Leibler divergence and of the Chi-Squared Divergence*

Description

KL.plugin computes the Kullback-Leiber (KL) divergence between two discrete random variables

x_1 and x_2. The corresponding probability mass functions are given by `freqs1` and `freqs2`. Note that the expectation is taken with regard to x_1 using `freqs1`.

`chi2.plugin` computes the chi-squared divergence between two discrete random variables x_1 and x_2 with `freqs1` and `freqs2` as corresponding probability mass functions. Note that the denominator contains `freqs2`.

Usage

```
KL.plugin(freqs1, freqs2, unit=c("log", "log2", "log10"))
chi2.plugin(freqs1, freqs2, unit=c("log", "log2", "log10"))
```

Arguments

`freqs1`	frequencies (probability mass function) for variable x_1.
`freqs2`	frequencies (probability mass function) for variable x_2.
`unit`	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set `unit="log2"`.

Details

Kullback-Leibler divergence between the two discrete variables x_1 to x_2 is $\sum_k p_1(k) \log(p_1(k)/p_2(k))$ where p_1 and p_2 are the probability mass functions of x_1 and x_2, respectively, and k is the index for the classes.

The chi-squared divergence is given by $\sum_k (p_1(k) - p_2(k))^2 / p_2(k)$.

Note that both the KL divergence and the chi-squared divergence are not symmetric in x_1 and x_2. The chi-squared divergence can be derived as a quadratic approximation of twice the KL divergence.

Value

`KL.plugin` returns the KL divergence.

`chi2.plugin` returns the chi-squared divergence.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

See Also

`KL.Dirichlet`, `KL.shrink`, `KL.empirical`, `mi.plugin`, `discretize2d`.

Examples

```
# load entropy library
library("entropy")

# probabilities for two random variables
freqs1 = c(1/5, 1/5, 3/5)
freqs2 = c(1/10, 4/10, 1/2)
```

```
# KL divergence between x1 to x2
KL.plugin(freqs1, freqs2)

# and corresponding (half) chi-squared divergence
0.5*chi2.plugin(freqs1, freqs2)

## relationship to Pearson chi-squared statistic

# Pearson chi-squared statistic and p-value
n = 30 # sample size (observed counts)
chisq.test(n*freqs1, p = freqs2) # built-in function

# Pearson chi-squared statistic from Pearson divergence
pcs.stat = n*chi2.plugin(freqs1, freqs2) # note factor n
pcs.stat

# and p-value
df = length(freqs1)-1 # degrees of freedom
pcs.pval = 1-pchisq(pcs.stat, df)
pcs.pval
```

`mi.plugin` *Plug-In Estimator of Mutual Information and of the Chi-Squared Statistic of Independence*

Description

`mi.plugin` computes the mutual information of two discrete random variables from the specified joint probability mass function.

`chi2indep.plugin` computes the chi-squared divergence of independence.

Usage

```
mi.plugin(freqs2d, unit=c("log", "log2", "log10"))
chi2indep.plugin(freqs2d, unit=c("log", "log2", "log10"))
```

Arguments

`freqs2d`	matrix of joint bin frequencies (joint probability mass function).
`unit`	the unit in which entropy is measured. The default is "nats" (natural units). For computing entropy in "bits" set `unit="log2"`.

Details

The mutual information of two random variables X and Y is the Kullback-Leibler divergence between the joint density/probability mass function and the product independence density of the marginals.

It can also defined using entropy as $MI = H(X) + H(Y) - H(X, Y)$.

Similarly, the chi-squared divergence of independence is the chi-squared divergence between the joint density and the product density. It is a second-order approximation of twice the mutual information.

Value

`mi.plugin` returns the mutual information.

`chi2indep.plugin` returns the chi-squared divergence of independence.

Author(s)

Korbinian Strimmer (https://strimmerlab.github.io).

See Also

`mi.Dirichlet`, `mi.shrink`, `mi.empirical`, `KL.plugin`, `discretize2d`.

Examples

```
# load entropy library
library("entropy")

# joint distribution of two discrete variables
freqs2d = rbind( c(0.2, 0.1, 0.15), c(0.1, 0.2, 0.25) )

# corresponding mutual information
mi.plugin(freqs2d)

# MI computed via entropy
H1 = entropy.plugin(rowSums(freqs2d))
H2 = entropy.plugin(colSums(freqs2d))
H12 = entropy.plugin(freqs2d)
H1+H2-H12

# and corresponding (half) chi-squared divergence of independence
0.5*chi2indep.plugin(freqs2d)
```

Index

∗ **univar**
discretize, 3
entropy, 5
entropy-package, 2
entropy.ChaoShen, 6
entropy.Dirichlet, 8
entropy.empirical, 11
entropy.MillerMadow, 13
entropy.NSB, 14
entropy.plugin, 16
entropy.shrink, 17
Gstat, 19
KL.plugin, 21
mi.plugin, 23

chi2.Dirichlet *(*entropy.Dirichlet*)*, 8
chi2.empirical *(*entropy.empirical*)*, 11
chi2.plugin, *2*, *12*, *20*
chi2.plugin *(*KL.plugin*)*, 21
chi2.shrink *(*entropy.shrink*)*, 17
chi2indep.Dirichlet
*(*entropy.Dirichlet*)*, 8
chi2indep.empirical
*(*entropy.empirical*)*, 11
chi2indep.plugin, *12*, *20*
chi2indep.plugin *(*mi.plugin*)*, 23
chi2indep.shrink *(*entropy.shrink*)*, 17
chi2stat, *2*, *12*
chi2stat *(*Gstat*)*, 19
chi2statindep, *12*
chi2statindep *(*Gstat*)*, 19

discretize, 3, *6*, *9*, *12*, *16*, *18*
discretize2d, *22*, *24*
discretize2d *(*discretize*)*, 3

entropy, *2*, *3*, 5, *7*, *9*, *12*, *15*, *16*, *18*
entropy-package, 2
entropy.ChaoShen, *5*, 6, *15*
entropy.Dirichlet, *5*, *7*, 8, *15*, *18*
entropy.empirical, *5*, *9*, 11, *14*, *16*
entropy.MillerMadow, *5*, 13
entropy.NSB, *5*, *7*, 14
entropy.plugin, *9*, *12*, 16, *18*
entropy.shrink, *2*, *5*, *7*, *9*, *15*, *16*, 17

freqs *(*entropy*)*, 5
freqs.Dirichlet *(*entropy.Dirichlet*)*, 8
freqs.empirical *(*entropy.empirical*)*, 11
freqs.shrink *(*entropy.shrink*)*, 17

Gstat, *2*, *12*, 19
Gstatindep, *12*
Gstatindep *(*Gstat*)*, 19

KL.Dirichlet, *22*
KL.Dirichlet *(*entropy.Dirichlet*)*, 8
KL.empirical, *22*
KL.empirical *(*entropy.empirical*)*, 11
KL.plugin, *2*, *9*, *12*, *16*, *18*, *20*, 21, *24*
KL.shrink, *22*
KL.shrink *(*entropy.shrink*)*, 17

mi.Dirichlet, *24*
mi.Dirichlet *(*entropy.Dirichlet*)*, 8
mi.empirical, *24*
mi.empirical *(*entropy.empirical*)*, 11
mi.plugin, *2*, *9*, *12*, *16*, *18*, *20*, *22*, 23
mi.shrink, *24*
mi.shrink *(*entropy.shrink*)*, 17