Package ‘TauStar’

March 21, 2017

Type Package

Title Efficient Computation and Testing of the Bergsma-Dassios Sign Covariance

Version 1.1.3

Date 2017-3-20

Maintainer Luca Weihs <lucaw@uw.edu>


License GPL (>= 3)

Imports Rcpp (>= 0.12.10)

LinkingTo Rcpp, RcppArmadillo

Suggests testthat

RoxygenNote 6.0.1

NeedsCompilation yes

Author Luca Weihs [aut, cre],
Emin Martinian [ctb] (Created the red-black tree library included in package.)

Repository CRAN

Date/Publication 2017-03-21 06:50:26 UTC

R topics documented:

- TauStar-package
- binaryQuantileSearch
- eigenForDiscreteProbs
- isDiscrete
Description

Computes the t* statistic corresponding to the tau star population coefficient introduced by Bergsma and Dassios (2014) <DOI:10.3150/13-BEJ514> and does so in O(n^2*log(n)) time following the algorithm of Weihs, Drton, and Leung (2016) <DOI:10.1007/s00180-015-0639-x>. Also allows for independence testing using the asymptotic distribution of t* as described by Nandy, Weihs, and Drton (2016) <http://arxiv.org/abs/1602.04387>. To directly compute the t* statistic see the function tStar. If otherwise interested in performing tests of independence then see the function tauStarTest.

Author(s)

Maintainer: Luca Weihs <lucaw@uw.edu>

Other contributors:

- Emin Martinian (Created the red-black tree library included in package.) [contributor]

References


**binaryQuantileSearch**

*Quantiles of a distribution.*

**Description**

Computes the pth quantile of a cumulative distribution function using a simple binary search algorithm. This can be extremely slow but has the benefit of being trivial to implement.

**Usage**

```r
binaryQuantileSearch(pDistFunc, p, lastLeft, lastRight, error = 10^-4)
```
Arguments

- **pDistFunc**: a cumulative distribution function on the real numbers, it should take a single argument \(x\) and return the cumulative distribution function evaluated at \(x\).
- **p**: the quantile \(p \in [0, 1]\)
- **lastLeft**: binary search works by continuously decreasing the search space from the left and right. lastLeft should be a lower bound for the quantile \(p\).
- **lastRight**: similar to lastRight but should be an upper bound.
- **error**: the error tolerated from the binary search

Value

the quantile (within error).

---

eigenForDiscreteProbs  
*Eigenvalues for discrete asymptotic distribution*

Description


Usage

```r
eigenForDiscreteProbs(p)
```

Arguments

- **p**: a vector of probabilities that sum to 1.

Value

the eigenvalues associated to the matrix generated by \(p\)
**isDiscrete**

Determine if input data is discrete

**Description**

Attempts to determine if the input data is from a discrete distribution. Will return true if the data type is of type integer or there are non-unique values.

**Usage**

isDiscrete(x)

**Arguments**

x  

a vector which should be determined if discrete or not.

**Value**

the best judgement of whether or not the data was discrete

---

**isProb**

Check if a Valid Probability

**Description**

Checks if the input vector has a single entry that is between 0 and 1

**Usage**

isProb(prob)

**Arguments**

prob  

the probability to check

**Value**

TRUE if conditions are met, FALSE if otherwise
isProbVector  

**Check if Vector of Probabilities**

**Description**
 Checks if the input vector has entries that sum to 1 and are non-negative

**Usage**
```r
isProbVector(probs)
```

**Arguments**
- `probs`  
  the probability vector to check

**Value**
- TRUE if conditions are met, FALSE if otherwise

isValidDataVector  

**Is Vector Valid Data?**

**Description**
 Determines if input vector is a valid vector of real valued observations

**Usage**
```r
isValidDataVector(x)
```

**Arguments**
- `x`  
  the vector to be tested

**Value**
- TRUE or FALSE
Null asymptotic distribution of $t^*$ in the discrete case

Description

Density, distribution function, quantile function and random generation for the asymptotic null distribution of $t^*$ in the discrete case. That is, in the case that $t^*$ is generated from a sample of jointly discrete independent random variables $X$ and $Y$.

Usage

```r
pDisHoeffInd(x, probs1, probs2, lower.tail = T, error = 10^-5)
dDisHoeffInd(x, probs1, probs2, error = 10^-3)
rDisHoeffInd(n, probs1, probs2)
qDisHoeffInd(p, probs1, probs2, error = 10^-4)
```

Arguments

- **x**: the value (or vector of values) at which to evaluate the function.
- **probs1**: a vector of probabilities corresponding to the (ordered) support of $X$. That is if your first random variable has support $u_1, ..., u_n$ then the $i$th entry of `probs` should be $\text{eqnP}(X = u_i)$.
- **probs2**: just as `probs1` but for the second random variable $Y$.
- **lower.tail**: a logical value, if TRUE (default), probabilities are $P(X \leq x)$ otherwise $P(X > x)$.
- **error**: a tolerated error in the result. This should be considered as a guide rather than an exact upper bound to the amount of error.
- **n**: the number of observations to return.
- **p**: the probability (or vector of probabilities) for which to get the quantile.

Value

$d\text{DisHoeffInd}$ gives the density, $p\text{DisHoeffInd}$ gives the distribution function, $q\text{DisHoeffInd}$ gives the quantile function, and $r\text{DisHoeffInd}$ generates random samples.
**pHoeffInd**

*Null asymptotic distribution of t* in the continuous case*

**Description**

Density, distribution function, quantile function and random generation for the asymptotic null distribution of \( t^* \) in the continuous case. That is, in the case that \( t^* \) is generated from a sample of jointly continuous independent random variables.

**Usage**

```r
pHoeffInd(x, lower.tail = T, error = 10^-5)

rHoeffInd(n)

dHoeffInd(x, error = 1/2 * 10^-3)

qHoeffInd(p, error = 10^-4)
```

**Arguments**

- **x**: the value (or vector of values) at which to evaluate the function.
- **lower.tail**: a logical value, if TRUE (default), probabilities are \( P(X \leq x) \) otherwise \( P(X > x) \).
- **error**: a tolerated error in the result. This should be considered as a guide rather than an exact upper bound to the amount of error.
- **n**: the number of observations to return.
- **p**: the probability (or vector of probabilities) for which to get the quantile.

**Value**

dHoeffInd gives the density, pHoeffInd gives the distribution function, qHoeffInd gives the quantile function, and rHoeffInd generates random samples.

**pmixHoeffInd**

*Null asymptotic distribution of t* in the mixed case*

**Description**

Density, distribution function, quantile function and random generation for the asymptotic null distribution of \( t^* \) in the mixed case. That is, in the case that \( t^* \) is generated a sample from an independent bivariate distribution where one coordinate is marginally discrete and the other marginally continuous.
print.ttest

Usage

\[
p_{\text{MixHoeffInd}}(x, \text{probs}, \text{lower.tail} = \text{T}, \text{error} = 10^{-6})
\]
\[
d_{\text{MixHoeffInd}}(x, \text{probs}, \text{error} = 10^{-3})
\]
\[
r_{\text{MixHoeffInd}}(n, \text{probs}, \text{error} = 10^{-8})
\]
\[
q_{\text{MixHoeffInd}}(\text{p}, \text{probs}, \text{error} = 10^{-4})
\]

Arguments

\begin{itemize}
\item **x**
  \text{the value (or vector of values) at which to evaluate the function.}
\item **probs**
  \text{a vector of probabilities corresponding to the (ordered) support the marginally discrete random variable. That is, if the marginally discrete distribution has support } u_1, ..., u_n \text{ then the } i \text{th entry of probs should be the probability of seeing } u_i.
\item **lower.tail**
  \text{a logical value, if TRUE (default), probabilities are } P(X \leq x) \text{ otherwise } P(X > x).
\item **error**
  \text{a tolerated error in the result. This should be considered as a guide rather than an exact upper bound to the amount of error.}
\item **n**
  \text{the number of observations to return.}
\item **p**
  \text{the probability (or vector of probabilities) for which to get the quantile.}
\end{itemize}

Value

\text{d}_{\text{MixHoeffInd}} \text{ gives the density, } p_{\text{MixHoeffInd}} \text{ gives the distribution function, } q_{\text{MixHoeffInd}} \text{ gives the quantile function, and } r_{\text{MixHoeffInd}} \text{ generates random samples.}

---

print.ttest  \hspace{1cm} \textit{Print Tau* Test Results}

Description

A simple print function for \texttt{ttest} (Tau* test) objects.

Usage

\[
## S3 method for class \texttt{ttest}'
\text{print}(x, \ldots)
\]

Arguments

\begin{itemize}
\item **x**
  \text{the \texttt{ttest} object to be printed}
\item **\ldots**
  \text{ignored.}
\end{itemize}
Description

Performs a (consistent) test of independence between two input vectors using the asymptotic (or permutation based) distribution of the test statistic $t^*$. The asymptotic results hold in the case that $x$ is generated from either a discrete or continuous distribution and similarly for $y$ (in particular it is allowed for one to be continuous while the other is discrete). The asymptotic distributions were computed in Nandy, Weihs, and Drton (2016) <http://arxiv.org/abs/1602.04387>.

Usage

tauStarTest(x, y, mode = "auto", resamples = 1000)

Arguments

- **x**: a vector of sampled values.
- **y**: a vector of sampled values corresponding to $x$, $y$ must be the same length as $x$.
- **mode**: should be one of five possible values: "auto", "continuous", "discrete", "mixed", or "permutation". If "auto" is selected then the function will attempt to automatically determine whether $x,y$ are discrete or continuous and then perform the appropriate asymptotic test. In cases "continuous", "discrete", and "mixed" we perform the associated asymptotic test making the given assumption. Finally if "permutation" is selected then the function runs a Monte-Carlo permutation test for some given number of resamplings.
- **resamples**: the number of resamplings to do if mode = "permutation". Otherwise this value is ignored.

Value

a list with class "tstest" recording the outcome of the test.

References


Examples

```r
set.seed(123)
x = rnorm(100)
y = rnorm(100)
testResults = tauStarTest(x,y)
print(testResults$pVal) # big p-value

y = y + x # make x and y correlated
```
tStar

Description
Computes the t* U-statistic for input data pairs \((x_1,y_1), (x_2,y_2), ..., (x_n,y_n)\) using the algorithm developed by Heller and Heller (2016) <arXiv:1605.08732> building off of the work of Weihs, Drton, and Leung (2015) <DOI:10.1007/s00180-015-0639-x>.

Usage
tStar(x, y, vStatistic = FALSE, resample = FALSE, numResamples = 500, sampleSize = min(length(x), 1000), method = "fastest", slow = FALSE)

Arguments
- **x**: A numeric vector of x values (length \(>= 4\)).
- **y**: A numeric vector of y values, should be of the same length as x.
- **vStatistic**: If TRUE then will compute the V-statistic version of t*, otherwise will compute the U-Statistic version of t*. Default is to compute the U-statistic.
- **resample**: If TRUE then will compute an approximation of t* using a subsetting approach: samples of size sampleSize are taken from the data numResample times, t* is computed on each subsample, and all subsample t* values are then averaged. Note that this only works when vStatistic == FALSE, in general you probably don’t want to compute the V-statistic via resampling as the size of the bias depends on the sampleSize irrespective numResamples. Default is resample == FALSE so that t* is computed on all of the data, this may be slow for very large sample sizes. Resampling can only be used when the method argument is using its default.
- **numResamples**: See resample variable description for details, this value is ignored if resample == FALSE (ignored by default).
- **sampleSize**: See resample variable description for details, this value is ignored if resample == FALSE (ignored by default).
- **method**: which method to use to compute the statistic. Default is "fastest" which uses the fastest available method (currently "heller"). The options are "heller" described in Heller and Heller (2016), "weihs", using the algorithm from Weihs et al. (2015), and "naive" using a naive algorithm.
- **slow**: a deprecated option kept for backwards compatibility. If TRUE then will override the method parameter and compute the t* statistic using a naive O(n^4) algorithm.

```r
testResults = tStarTest(x, y)
print(testResults$pVal) # small p-value
```
Value

The numeric value of the t* statistic.

References


Examples

```r
## Not run:
library(TauStar)

tStar(c(1,2,3,4), c(1,2,3,4)) # == 2/3

tStar(c(1,2,3,4), c(1,-1,-1,-1)) # == -1/3

set.seed(23421)
tStar(rnorm(4000), rnorm(4000)) # near 0

set.seed(923)
tStar(rnorm(100), rnorm(100), vStatistic = TRUE)

set.seed(9492)
tStar(rnorm(10000), rnorm(10000), resample = TRUE, sampleSize = 30, numResamples = 5000)
```

## End(Not run)
Index

binaryQuantileSearch, 3

dDisHoeffInd (pDisHoeffInd), 7
dHoeffInd (pHoeffInd), 8
dMixHoeffInd (pMixHoeffInd), 8
eigenForDiscreteProbs, 4

isDiscrete, 5
isProb, 5
isProbVector, 6
isValidDataVector, 6

pDisHoeffInd, 7
pHoeffInd, 8
pMixHoeffInd, 8
print.tstest, 9

qDisHoeffInd (pDisHoeffInd), 7
qHoeffInd (pHoeffInd), 8
qMixHoeffInd (pMixHoeffInd), 8

rDisHoeffInd (pDisHoeffInd), 7
rHoeffInd (pHoeffInd), 8
rMixHoeffInd (pMixHoeffInd), 8

TauStar (TauStar-package), 2
TauStar-package, 2
tauStarTest, 10
tStar, 11