Package 'eummd'

October 2, 2023

Title Efficient Univariate Maximum Mean Discrepancy

Version 0.1.4

Date 2023-10-02

Description Computes maximum mean discrepancy two-sample test for univariate data using the Laplacian kernel, as described in Bodenham and Kawahara (2023) <doi:10.1007/s11222-023-10271-x>. The p-value is computed using permutations. Also includes implementation for computing the robust median difference statistic 'Q_n' from Croux and Rousseeuw (1992) <doi:10.1007/978-3-662-26811-7_58> based on Johnson and Mizoguchi (1978) <doi:10.1137/0207013>.

Depends R (>= 4.1.0), Rcpp (>= 1.0.0)

License GPL-2 | GPL-3

Encoding UTF-8

RoxygenNote 7.2.3

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

LinkingTo Rcpp

NeedsCompilation yes

Author Dean Bodenham [aut, cre]

Maintainer Dean Bodenham <deanbodenhampkgs@gmail.com>

Repository CRAN

Date/Publication 2023-10-02 12:30:02 UTC

R topics documented:

`energydist` *Naive computation for Energy Distance*

Description

Computes energy distance, and possibly a p-value. Suitable for multivariate data. Naive approach, quadratic in number of observations.

Usage

```
energydist(X, Y, pval = TRUE, numperm = 200, seednum = 0)
```

Arguments

`X`	Matrix (or vector) of observations in first sample.
`Y`	Matrix (or vector) of observations in second sample.
`pval`	Boolean for whether to compute p-value or not.
`numperm`	Number of permutations. Default is `200`.
`seednum`	Seed number for generating permutations. Default is `0`, which means seed is set randomly. For values larger than `0`, results will be reproducible.

Details

First checks number of columns (dimension) are equal. Suppose matrix X has n rows and d columns, and matrix Y has m rows; checks that Y has d columns (if not, then throws error). Then flattens matrices to vectors (or, if $d = 1$, they are already vectors. Then calls C++ method. If the first sample has n d-dimensional samples and the second sample has m d-dimensional samples, then the algorithm computes the statistic in $O((n+m)^2)$ time.

Random seed is set for `std::mt19937` and `std::shuffle` in C++.

Value

A list with the following elements:

`pval` The p-value of the test, if it is computed (`pval=TRUE`).

`stat` The statistic of the test, which is always computed.

References

Baringhaus L. and Franz C. (2004) "On a new multivariate two-sample test." Journal of multivariate analysis 88(1):190-206

Szekely G. J. and Rizzo M. L. (2004) "Testing for equal distributions in high dimension." InterStat 5(16.10):1249-1272

Examples

```
X <- matrix(c(1:12), ncol=2, byrow=TRUE)
Y <- matrix(c(13:20), ncol=2, byrow=TRUE)
energydistList <- energydist(X=X, Y=Y, pval=FALSE)

#computing p-value
energydistList <- energydist(X=X, Y=Y)

#computing p-value
#using 1000 permutations and seed 1 for reproducibility.
energydistList <- energydist(X=X, Y=Y, numperm=1000, seednum=1)
```

---

eummd *euMMD: Efficient Univariate Maximum Mean Discrepancy*

Description

Computes the maximum mean discrepancy statistic with the Laplacian kernel. Suitable only for univariate data. Computing the statistic alone for n observations is $O(n \log n)$, and computing the p-value for L permutations is $O(n \log n + Ln)$.

Usage

```
eummd(x, y, beta = -0.1, pval = TRUE, numperm = 200, seednum = 0)
```

Arguments

`x`	Univariate vector of observations in first sample.
`y`	Univariate vector of observations in second sample.
`beta`	kernel parameter. Must be positive; if not, computes median heuristic in quadratic time. Default value is `-0.1`, which will force median heuristic to be used.
`pval`	Boolean for whether to compute p-value or not.
`numperm`	Number of permutations. Default is `200`.
`seednum`	Seed number for generating permutations. Default is `0`, which means seed is set randomly. For values larger than `0`, results will be reproducible.

Details

If the total number of observations in both samples is `n`, first sort combined sample in $O(n \log n)$ before remaining steps are linear in `n`.

If `beta` is not a positive value, median difference is computed as follows:

$$m = \text{median}\{||x_i - x_j||_1;\ i > j,\ i = 1, 2, \ldots, n + m, \ \text{ and } j = 1, 2, \ldots, i - 1\},$$

where $||x_i - x_j||_1$ is the 1-norm, and so if the data are univariate then

$$||x_i - x_j||_1 = |x_i - x_j|.$$

and finally median heuristic is `beta = 1/m`. This can be computed in $O(n \log n)$ time using the algorithms of Johnson and Mizoguchi (1978) and Croux and Rousseuw (1992); see `mediandiff` for references.

The Laplacian kernel k is defined as

$$k(x, y) = \exp[-\beta||x - y||_1].$$

The random seed is set for `std::mt19937` and `std::shuffle` in C++.

Value

A list with the following elements:

`pval` The p-value of the test, if it is computed (`pval=TRUE`). Otherwise, it is set to `NA`.

`stat` The statistic of the test, which is always computed.

`beta` The kernel parameter used in the test. If `beta` was not initialised or negative, this will be the median heuristic value.

References

Bodenham, D. A., and Kawahara, Y. (2023) "euMMD: efficiently computing the MMD two-sample test statistic for univariate data." Statistics and Computing 33.5 (2023): 110.

Croux, C. and Rousseeuw, P. J. (1992), "Time-Efficient Algorithms for Two Highly Robust Estimators of Scale" In Computational Statistics: Volume 1: Proceedings of the 10th Symposium on Computational Statistics (pp. 411-428).

Johnson, D.B., and Mizoguchi, T. (1978), "Selecting the Kth Element in X + Y and X_1 + X_2 + ... + X_m", SIAM Journal of Computing, 7, 147-153.

See Also

`mediandiff`

Examples

```
x <- c(7.1, 1.2, 4.3, 0.4)
y <- c(5.5, 2.6, 8.7)
#setting the kernel parameter to be 0.1; setting seed=1 for reproducibility
mmd_list <- eummd(x, y, beta=0.1, seednum=1)

#now using median heuristic (default)
mmd_list <- eummd(x, y, seednum=1)

#now not computing the p-value, only the statistic
mmd_list <- eummd(x, y, pval=FALSE, seednum=1)
```

```
#now using a larger number of permutations
mmd_list <- eummd(x, y, numperm=1000, seednum=1)
```

---

meammd *MEA-MMD: Multivariate Efficient Approximate Maximum Mean Discrepancy*

Description

Computes maximum mean discrepancy statistics with Laplacian or Gaussian kernel. Suitable for multivariate data. Naive approach, quadratic in number of observations.

Usage

```
meammd(
  X,
  Y,
  beta = -0.1,
  pval = TRUE,
  type = c("proj", "dist"),
  numproj = 20,
  nmethod = c(2, 1),
  distpval = c("Hommel", "Fisher"),
  numperm = 200,
  seednum = 0
)
```

Arguments

X	Matrix (or vector) of observations in first sample.
Y	Matrix (or vector) of observations in second sample.
beta	kernel parameter. Must be positive; if not, computes median heuristic in quadratic time for each projection. Default value is -0.1, which will force median heuristic to be used.
pval	Boolean for whether to compute p-value or not.
type	The type of projection used. Either "proj" for random projections (default) or "dist" for interpoint distances.
numproj	Number of projections (only used if type="proj"). Default is 20.
nmethod	Norm used for interpoint distances, if type="dist". Needs to be either 2 (for two-norm, default) or 1 (for one-norm).
distpval	The p-value combination procedure if type="dist". Options are "Hommel" (default) or "Fisher". The Hommel method is preferred since the Type I error does not seem to be controlled if the Fisher method is used.

numperm	Number of permutations. Default is 200.
seednum	Seed number for generating permutations. Default is 0, which means seed is set randomly. For values larger than 0, results will be reproducible.

Value

A list with the following elements:

pval The p-value of the test, if it is computed (pval=TRUE). Otherwise, it is set to NA.

stat The statistic of the test, which is only returned when type="proj", otherwise it is set to NA.

References

Bodenham, D. A., and Kawahara, Y. (2023) "euMMD: efficiently computing the MMD two-sample test statistic for univariate data." Statistics and Computing 33.5 (2023): 110.

Examples

```
X <- matrix(c(1:12), ncol=2, byrow=TRUE)
Y <- matrix(c(13:20), ncol=2, byrow=TRUE)
# using the random projections method
mmdList <- meammd(X=X, Y=Y, pval=TRUE, type="proj", numproj=50)

# using the method were distances are computed to the various points
mmdList <- meammd(X=X, Y=Y, pval=TRUE, type="dist")
```

---

mediandiff *Compute the median difference between pairs of values*

Description

Compute the median of all differences between distinct pairs in vectors or matrices.

Usage

```
mediandiff(X, Y = NULL, kernel = c("Laplacian", "Gaussian"), fast = FALSE)
```

Arguments

X	Numeric vector or matrix of length n.
Y	Numeric vector or matrix of length m, or NULL.
kernel	String, either "Laplacian" or "Gaussian".
fast	Boolean; if TRUE will run $O(N \log N)$ algorithm, where N = n + m, but if FALSE (default) will run naive $O(N^2 \log N)$ algorithm.

Details

The median difference is defined as follows:

Z is the combined X and Y values into a single vector or matrix. Number of columns is the dimension, and these need to be equal for X and Y. Then, if the Laplacian kernel is used,

$$m = \text{median}\{||x_i - x_j||_1;\ i > j,\ i = 1, 2, \ldots, n + m,\ \text{ and } j = 1, 2, \ldots, i - 1\},$$

where $||z_i - z_j||_1$ is the 1-norm, and so if the data are d-dimensional then

$$||z_i - z_j||_1 = \sum_{k=1}^{d} |z_{i,k} - z_{j,k}|.$$

If the Gaussian kernel is specified, then the square of the two-norm is used.

The median heuristic is defined as `beta = 1/m`.

Naive method will compute all distinct pairs, of which there are $N(N + 1)/2$ differences. These are then sorted using a $O(N \log N)$ algorithm, so overall $O(N^2 \log N)$.

The fast method is $O(N \log N)$ is from Croux and Rousseeuw (1992), which is based on Johnson and Mizoguchi (1978).

Value

A scalar, the median of all pairwise differences.

References

Croux, C. and Rousseeuw, P. J. (1992), "Time-Efficient Algorithms for Two Highly Robust Estimators of Scale" In Computational Statistics: Volume 1: Proceedings of the 10th Symposium on Computational Statistics (pp. 411-428).

Johnson, D.B., and Mizoguchi, T. (1978), "Selecting the Kth Element in X + Y and X_1 + X_2 + ... + X_m", SIAM Journal of Computing, 7, 147-153.

See Also

`medianheuristic`

Examples

```
X <- c(7.1, 1.2, 4.3, 0.4)
Y <- c(5.5, 2.6, 8.7)
#using fast method, Laplacian kernel, loglinear in number of observations
md <- mediandiff(X, Y, fast=TRUE)

#using fast method, Gaussian kernel, loglinear in number of observations
md <- mediandiff(X, Y, fast=TRUE, kernel="Gaussian")

#using naive method (default), with Laplacian kernel
md <- mediandiff(X, Y)
```

`medianheuristic` *Compute the median heuristic*

Description

Computes the inverse of the median difference of all distinct pairs in vectors or matrices.

Usage

```
medianheuristic(X, Y = NULL, kernel = c("Laplacian", "Gaussian"), fast = FALSE)
```

Arguments

`X`	Numeric vector or matrix of length `n`.
`Y`	Numeric vector or matrix of length `m`, or `NULL`.
`kernel`	String, either `"Laplacian"` or `"Gaussian"`.
`fast`	Boolean; if `TRUE` will run `O(N log N)` algorithm, where `N = n + m`, but if `FALSE` will run naive `O(N^2 log(N))` algorithm.

Details

Computes median of differences `md` using `mediandiff` and then returns `1 / md`. See `mediandiff` for details.

Value

A scalar, the inverse of the median of all pairwise differences.

See Also

`mediandiff`

Examples

```
X <- c(7.1, 1.2, 4.3, 0.4)
Y <- c(5.5, 2.6, 8.7)
mh <- medianheuristic(X, Y, kernel="Laplacian", fast=TRUE)

#using fast method, Gaussian kernel, loglinear in number of observations
mh <- medianheuristic(X, Y, fast=TRUE, kernel="Gaussian")

#using naive method (default), with Laplacian kernel
mh <- medianheuristic(X, Y)
```

`mmd` *Naive computation for Maximum Mean Discrepancy*

Description

Computes maximum mean discrepancy statistics with Laplacian or Gaussian kernel. Suitable for multivariate data. Naive approach, quadratic in number of observations.

Usage

```
mmd(
  X,
  Y,
  beta = -0.1,
  pval = TRUE,
  kernel = c("Laplacian", "Gaussian"),
  numperm = 200,
  seednum = 0
)
```

Arguments

`X`	Matrix (or vector) of observations in first sample.
`Y`	Matrix (or vector) of observations in second sample.
`beta`	kernel parameter. Must be positive; if not, computes median heuristic in quadratic time. Default value is `-0.1`, which will force median heuristic to be used.
`pval`	Boolean for whether to compute p-value or not.
`kernel`	String, either `"Laplacian"` or `"Gaussian"`. Default is `"Laplacian"`.
`numperm`	Number of permutations. Default is `200`.
`seednum`	Seed number for generating permutations. Default is `0`, which means seed is set randomly. For values larger than `0`, results will be reproducible.

Details

First checks number of columns (dimension) are equal. Suppose matrix X has n rows and d columns, and matrix Y has m rows; checks that Y has d columns (if not, then throws error). Then flattens matrices to vectors (or, if $d = 1$, they are already vectors. Then calls C++ method. If the first sample has n d-dimensional samples and the second sample has m d-dimensional samples, then the algorithm computes the statistic in $O((n+m)^2)$ time.

Median difference is as follows:

$$m = \text{median}\{||x_i - x_j||_1;\ i > j,\ i = 1, 2, \ldots, n+m, \quad \text{and } j = 1, 2, \ldots, i-1\},$$

where $||x_i - x_j||_1$ is the 1-norm, and so if the data are d-dimensional then

$$||x_i - x_j||_1 = \sum_{k=1}^{d} |x_{i,k} - x_{j,k}|,$$

and finally median heuristic is `beta = 1/m`. This can be computed in $O((n+m)^2)$ time.

The Laplacian kernel k is defined as

$$k(x, y) = \exp(-\beta||x_i - x_j||_1).$$

Random seed is set for `std::mt19937` and `std::shuffle` in C++.

Value

A list with the following elements:

`pval` The p-value of the test, if it is computed (`pval=TRUE`).

`stat` The statistic of the test, which is always computed.

`beta` The kernel parameter used in the test. If `beta` was not initialised or negative, this will be the median heuristic value.

References

Gretton, A., Borgwardt, K. M., Rasch M. J., Schölkopf, B. and Smola, A. (2012) "A kernel two-sample test." The Journal of Machine Learning Research 13, no. 1, 723-773.

Examples

```
X <- matrix(c(1:12), ncol=2, byrow=TRUE)
Y <- matrix(c(13:20), ncol=2, byrow=TRUE)
mmdList <- mmd(X=X, Y=Y, beta=0.1, pval=FALSE)

#using median heuristic
mmdList <- mmd(X=X, Y=Y, pval=FALSE)

#using median heuristic and computing p-value
mmdList <- mmd(X=X, Y=Y)

#using median heuristic and computing p-value
#using 1000 permutations and seed 1 for reproducibility.
mmdList <- mmd(X=X, Y=Y, numperm=1000, seednum=1)
```

Index