Package 'robslopes'

February 24, 2022

Type Package

Title Fast Algorithms for Robust Slopes

Version 1.1.0

Date 2022-02-23

Author Jakob Raymaekers

Maintainer Jakob Raymaekers <j.raymaekers@maastrichtuniversity.nl>

Description Fast algorithms for the Theil-Sen estimator,
Siegel's repeated median slope estimator, and Passing-Bablok regression.
The implementation is based on algorithms by
Dillencourt et. al (1992) <doi:10.1142/S0218195992000020> and Matousek et. al (1998) <doi:10.1007/PL00009190>.
The implementation of Passing-Bablok regression is explained in detail in
Raymaekers J., Dufey F. (2022). Equivariant Passing-Bablok regression in quasilinear time. <arXiv:2202.08060>.
All algorithms run in quasilinear time.

License GPL (>= 2)

Imports Rcpp (>= 1.0.5)

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-02-24 12:00:02 UTC

R topics documented:

PassingBablok *Passing-Bablok slope and intercept estimator.*

Description

Computes the equivariant Passing-Bablok regression. The implemented algorithm was proposed by Raymaekers and Dufey (2022) and runs in an expected $O(nlogn)$ time while requiring $O(n)$ storage.

Usage

```
PassingBablok(x, y, alpha = NULL, verbose = TRUE)
```

Arguments

x	A vector of predictor values.
y	A vector of response values.
alpha	Determines the order statistic of the target slope, which is equal to $[alpha * n * (n-1)]$, where n denotes the sample size. Defaults to NULL, which corresponds with the (upper) median.
verbose	Whether or not to print out the progress of the algorithm. Defaults to TRUE.

Details

Given two input vectors x and y of length n, the equivariant Passing-Bablok estimator is computed as $med_{ij}|(y_i - y_j)/(x_i - x_j)|$. By default, the median in this experssion is the upper median, defined as $\lfloor (n+2)/2 \rfloor$. By changing alpha, other order statistics of the slopes can be computed.

Value

A list with elements:

intecept	The estimate of the intercept.
slope	The Theil-Sen estimate of the slope.

Author(s)

Jakob Raymaekers

References

Passing, H., Bablok, W. (1983). A new biometrical procedure for testing the equality of measurements from two different analytical methods. Application of linear regression procedures for method comparison studies in clinical chemistry, Part I, *Journal of clinical chemistry and clinical biochemistry*, **21**,709-720.

Bablok, W., Passing, H., Bender, R., Schneider, B. (1988). A general regression procedure for method transformation. Application of linear regression procedures for method comparison studies in clinical chemistry, Part III. *Journal of clinical chemistry and clinical biochemistry*, **26**,783-790.

Raymaekers J., Dufey F. (2022). Equivariant Passing-Bablok regression in quasilinear time. (link to open access pdf)

Examples

```
# We compare the implemented algorithm against a naive brute-force approach.

bruteForcePB <- function(x, y) {

  n <- length(x)
  medind1 <- floor(((n * (n - 1)) / 2 + 2) / 2) # upper median
  medind2 <- floor((n + 2) / 2)
  temp <-  t(sapply(1:n, function(z)  apply(cbind(x, y), 1 ,
                                            function(k) (k[2] - y[z]) /
                                              (k[1] - x[z]))))
  PBslope <- sort(abs(as.vector(temp[lower.tri(temp)])))[medind1]
  PBintercept <- sort(y - x * PBslope)[medind2]
  return(list(intercept = PBintercept, slope = PBslope))
}


n = 100
set.seed(2)
x = rnorm(n)
y = x + rnorm(n)

t0 <- proc.time()
PB.fast <- PassingBablok(x, y, NULL, FALSE)
t1 <- proc.time()
t1 - t0

t0 <- proc.time()
PB.naive <- bruteForcePB(x, y)
t1 <- proc.time()
t1 - t0

PB.fast$slope - PB.naive$slope
```

`RepeatedMedian` *Siegel's repeated median slope and intercept estimator.*

Description

Computes the repeated median slope proposed by Siegel (1982) using the algorithm by Matousek et. al (1998). The algorithm runs in an expected $O(n(logn)^2)$ time, which is typically significantly faster than the $O(n^2)$ computational cost of the naive algorithm, and requires $O(n)$ storage.

Usage

```
RepeatedMedian(x, y, alpha = NULL, beta = NULL, verbose = TRUE)
```

Arguments

x	A vector of predictor values.
y	A vector of response values.
alpha	Determines the outer order statistic, which is equal to $[alpha * n]$, where n denotes the sample size. Defaults to NULL, which corresponds with the (upper) median.
beta	Determines the inner order statistic, which is equal to $[beta * (n - 1)]$, where n denotes the sample size. Defaults to NULL, which corresponds with the (upper) median.
verbose	Whether or not to print out the progress of the algorithm. Defaults to TRUE.

Details

Given two input vectors x and y of length n, the repeated median is computed as $med_i med_j (y_i - y_j)/(x_i - x_j)$. The default "outer" median is the $\lfloor (n+2)/2 \rfloor$ largest element in the ordered median slopes. The inner median, which for each observation is calculated as the median of the slopes connected to this observation, is the $\lfloor (n+1)/2 \rfloor$ largest element in the ordered slopes. By changing alpha and beta, other repeated order statistics of the slopes can be computed.

Value

A list with elements:

intecept	The estimate of the intercept.
slope	The Theil-Sen estimate of the slope.

Author(s)

Jakob Raymaekers

References

Siegel, A. F. (1982). Robust regression using repeated medians. *Biometrika*, **69**(1), 242-244.

Matousek, J., Mount, D. M., & Netanyahu, N. S. (1998). Efficient randomized algorithms for the repeated median line estimator. *Algorithmica*, **20**(2), 136-150.

See Also

TheilSen

Examples

```
# We compare the implemented algorithm against a naive brute-force approach.

bruteForceRM <- function(x, y) {

  n <- length(x)
  medind1 <- floor((n+2) / 2)
  medind2 <- floor((n+1) / 2)
  temp <-  t(sapply(1:n, function(z)  sort(apply(cbind(x, y), 1 ,
                                                 function(k) (k[2] - y[z]) /
                                                   (k[1] - x[z])))))
  RMslope <- sort(temp[, medind2])[medind1]
  RMintercept <- sort(y - x * RMslope)[medind1]
  return(list(intercept = RMintercept, slope = RMslope))
}

n = 100
set.seed(2)
x = rnorm(n)
y = x + rnorm(n)

t0 <- proc.time()
RM.fast <- RepeatedMedian(x, y, NULL, NULL, FALSE)
t1 <- proc.time()
t1 - t0

t0 <- proc.time()
RM.naive <- bruteForceRM(x, y)
t1 <- proc.time()
t1 - t0

RM.fast$slope - RM.naive$slope
```

`TheilSen` *Theil-Sen slope and intercept estimator.*

Description

Computes the Theil-Sen median slope estimator by Theil (1950) and Sen (1968). The implemented algorithm was proposed by Dillencourt et. al (1992) and runs in an expected $O(nlogn)$ time while requiring $O(n)$ storage.

Usage

```
TheilSen(x, y, alpha = NULL, verbose = TRUE)
```

Arguments

x	A vector of predictor values.
y	A vector of response values.
alpha	Determines the order statistic of the target slope, which is equal to $[alpha * n * (n-1)]$, where n denotes the sample size. Defaults to NULL, which corresponds with the (upper) median.
verbose	Whether or not to print out the progress of the algorithm. Defaults to TRUE.

Details

Given two input vectors x and y of length n, the Theil-Sen estimator is computed as $med_{ij}(y_i - y_j)/(x_i - x_j)$. By default, the median in this experssion is the upper median, defined as $\lfloor (n+2)/2 \rfloor$. By changing alpha, other order statistics of the slopes can be computed.

Value

A list with elements:

intecept	The estimate of the intercept.
slope	The Theil-Sen estimate of the slope.

Author(s)

Jakob Raymaekers

References

Theil, H. (1950), A rank-invariant method of linear and polynomial regression analysis (Parts 1-3), *Ned. Akad. Wetensch. Proc. Ser. A*, **53**, 386-392, 521-525, 1397-1412.

Sen, P. K. (1968). Estimates of the regression coefficient based on Kendall's tau. *Journal of the American statistical association*, **63**(324), 1379-1389.

Dillencourt, M. B., Mount, D. M., & Netanyahu, N. S. (1992). A randomized algorithm for slope selection. *International Journal of Computational Geometry & Applications*, **2**(01), 1-27.

Examples

```
# We compare the implemented algorithm against a naive brute-force approach.

bruteForceTS <- function(x, y) {

  n <- length(x)
  medind1 <- floor(((n * (n - 1)) / 2 + 2) / 2)
  medind2 <- floor((n + 2) / 2)
  temp <-  t(sapply(1:n, function(z)  apply(cbind(x, y), 1 ,
                                            function(k) (k[2] - y[z]) /
                                              (k[1] - x[z]))))
  TSslope <- sort(as.vector(temp[lower.tri(temp)]))[medind1]
  TSintercept <- sort(y - x * TSslope)[medind2]
  return(list(intercept = TSintercept, slope = TSslope))
```

```
}


n = 100
set.seed(2)
x = rnorm(n)
y = x + rnorm(n)

t0 <- proc.time()
TS.fast <- TheilSen(x, y, NULL, FALSE)
t1 <- proc.time()
t1 - t0

t0 <- proc.time()
TS.naive <- bruteForceTS(x, y)
t1 <- proc.time()
t1 - t0

TS.fast$slope - TS.naive$slope
```

Index