Package ‘kernelboot’

February 14, 2020

Type Package
Title Smoothed Bootstrap and Random Generation from Kernel Densities
Version 0.1.7
Date 2020-02-13
Author Tymoteusz Wołodzko
Maintainer Tymoteusz Wołodzko <twolodzko+kernelboot@gmail.com>
Description Smoothed bootstrap and functions for random generation from univariate and multivariate kernel densities. It does not estimate kernel densities.
License GPL-2
URL https://github.com/twolodzko/kernelboot
BugReports https://github.com/twolodzko/kernelboot/issues
Depends R (>= 3.1.0)
LinkingTo Rcpp
Imports Rcpp, future, future.apply
Suggests covr, testthat, ks, KernSmooth, cramer
Encoding UTF-8
LazyData TRUE
RoxygenNote 7.0.2
NeedsCompilation yes
Repository CRAN
Date/Publication 2020-02-13 23:10:03 UTC

R topics documented:

  bw.silv ................................................................. 2
  kernelboot ........................................................... 3
  kernelboot-class .................................................... 8
  rmvg ................................................................. 8
bw.silv

Bandwidth selector for multivariate kernel density estimation

Description

Rule of thumb bandwidth selectors for Gaussian kernels as described by Scott (1992) and Silverman (1986).

Usage

bw.silv(x, na.rm = FALSE)

bw.scott(x, na.rm = FALSE)

Arguments

x numeric matrix or data.frame.

na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.

Details

Scott’s (1992) rule is defined as

\[ H = n^{-2/(m+4)} \hat{\Sigma} \]

Silverman’s (1986; see Chacon, Duong and Wand, 2011) rule is defined as

\[ H = \left( \frac{4}{n(m+2)} \right)^{2/(m+4)} \hat{\Sigma} \]

where \( m \) is number of variables, \( n \) is sample size, \( \hat{\Sigma} \) is the empirical covariance matrix. The bandwidth is returned as a covariance matrix, so to use it for a product kernel, take square root of it’s diagonal: sqrt(diag(H)).

bw.silv corresponds to Hns method with deriv.order=0 from the ks package.
References


See Also

bandwidth

kernelboot  Smoothed bootstrap

Description

Smoothed bootstrap is an extension of standard bootstrap using kernel densities.

Usage

kernelboot(
  data,
  statistic,
  R = 500L,
  bw = "default",
  kernel = c("multivariate", "gaussian", "epanechnikov", "rectangular", "triangular",
            "biweight", "cosine", "optcosine", "none"),
  weights = NULL,
  adjust = 1,
  shrinked = TRUE,
  ignore = NULL,
  parallel = FALSE,
  workers = 1L
)

Arguments

data  vector, matrix, or data.frame. For non-numeric values standard bootstrap is applied (see below).
statistic  a function that is applied to the data. The first argument of the function will always be the original data.
R  the number of bootstrap replicates.
the smoothing bandwidth to be used (see density). The kernels are scaled such that this is the standard deviation, or covariance matrix of the smoothing kernel. By default bw.nrd0 is used for univariate data, and bw.silver is used for multivariate data. When using kernel = "multivariate" this parameter should be a covariance matrix of the smoothing kernel.

kernel

a character string giving the smoothing kernel to be used. This must partially match one of "multivariate", "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine", "optcosine", or "none" with default "multivariate", and may be abbreviated. Using kernel = "multivariate" forces multivariate Gaussian kernel (or univariate Gaussian for univariate data). Using kernel = "none" forces using standard bootstrap (no kernel smoothing).

weights

vector of importance weights. It should have as many elements as there are observations in data. It defaults to uniform weights.

adjust

scalar; the bandwidth used is actually adjust*bw. This makes it easy to specify values like 'half the default' bandwidth.

shrinked

logical; if TRUE random generation algorithm preserves means and variances of the variables. This parameter is ignored for "multivariate" kernel.

ignore

vector of names of columns to be ignored during the smoothing phase of bootstrap procedure (their values are not altered using random noise).

parallel

if TRUE, parallel computing is used (see future_lapply). Warning: using parallel computing does not necessary have to lead to improved performance.

workers

the number of workers used for parallel computing (see multiprocess).

Details

Smoothed bootstrap is an extension of standard bootstrap procedure, where instead of drawing samples with replacement from the empirical distribution, they are drawn from kernel density estimate of the distribution.

For smoothed bootstrap, points (in univariate case), or rows (in multivariate case), are drawn with replacement, to obtain samples of size \( n \) from the initial dataset of size \( n \), as with standard bootstrap. Next, random noise from kernel density \( K \) is added to each of the drawn values. The procedure is repeated \( R \) times and statistic is evaluated on each of the samples.

The noise is added only to the numeric columns, while non-numeric columns (e.g. character, factor, logical) are not altered. What follows, to the non-numeric columns and columns listed in ignore parameter standard bootstrap procedure is applied.

Univariate kernel densities

Univariate kernel density estimator is defined as

\[
\hat{f}_h(x) = \sum_{i=1}^{n} w_i K_h(x - y_i)
\]

where \( w \) is a vector of weights such that all \( w_i \geq 0 \) and \( \sum w_i = 1 \) (by default uniform \( 1/n \) weights are used), \( K_h = K(x/h)/h \) is kernel \( K \) parametrized by bandwidth \( h \) and \( y \) is a vector of data points used for estimating the kernel density.
To draw samples from univariate kernel density, the following procedure can be applied (Silverman, 1986):

**Step 1** Sample $i$ uniformly with replacement from $1, \ldots, n$.

**Step 2** Generate $\varepsilon$ to have probability density $K$.

**Step 3** Set $x = y_i + h\varepsilon$.

If samples are required to have the same variance as data (i.e. shrinked = TRUE), then **Step 3** is modified as following:

**Step 3’** $x = \bar{y} + (y_i - \bar{y} + h\varepsilon)/(1 + h^2\sigma^2_K/\sigma^2_Y)^{1/2}$

where $\sigma^2_K$ is variance of the kernel (fixed to 1 for kernels used in this package).

When shrinkage described in **Step 3’** is applied, the smoothed bootstrap density function changes it’s form to

$$\hat{f}_{h,b}(x) = (1 + r) \hat{f}_h(x + r(x - \bar{y}))$$

where $r = (1 + h^2\sigma^2_K/\sigma^2_Y)^{1/2} - 1$.

This package offers the following univariate kernels:

- **Gaussian** $\frac{1}{\sqrt{2\pi}} e^{-u^2/2}$
- **Rectangular** $\frac{1}{2} 1_{(|u|\leq 1)}$
- **Triangular** $(1 - |u|) 1_{(|u|\leq 1)}$
- **Epanchenikov** $\frac{3}{4} (1 - u^2) 1_{(|u|\leq 1)}$
- **Biweight** $\frac{15}{16} (1 - u^2)^2 1_{(|u|\leq 1)}$
- **Cosine** $\frac{3}{2} (1 + \cos(\pi u)) 1_{(|u|\leq 1)}$
- **Optcosine** $\frac{2}{4} \cos \left( \frac{\pi}{2} u \right) 1_{(|u|\leq 1)}$

All the kernels are re-scaled so that their standard deviations are equal to 1, so that bandwidth parameter controls their standard deviations.

Random generation from Epanchenikov kernel is done using algorithm described by Devroye (1986). For optcosine kernel inverse transform sampling is used. For biweight kernel random values are drawn from Beta(3, 3) distribution and Beta(3.3575, 3.3575) distribution serves as a close approximation of cosine kernel. Random generation for triangular kernel is done by taking difference of two i.i.d. uniform random variates. To sample from rectangular and Gaussian kernels standard random generation algorithms are used (see runif and rnorm).

**Product kernel densities**

Univariate kernels may easily be extended to multiple dimensions by using product kernel

$$\hat{f}_H(x) = \sum_{i=1}^n w_i \prod_{j=1}^m K_{h_j}(x_i - y_{ij})$$

where $w$ is a vector of weights such that all $w_i \geq 0$ and $\sum_i w_i = 1$ (by default uniform $1/n$ weights are used), and $K_{h_j}$ are univariate kernels $K$ parametrized by bandwidth $h_j$, where $y$ is a matrix of data points used for estimating the kernel density.
Random generation from product kernel is done by drawing with replacement rows of \( y_i \), and then adding to the sampled values random noise from univariate kernels \( K \), parametrized by corresponding bandwidth parameters \( h_j \).

**Multivariate kernel densities**

Multivariate kernel density estimator may also be defined in terms of multivariate kernels \( K_H \) (e.g., multivariate normal distribution, as in this package)

\[
\hat{f}_H(x) = \sum_{i=1}^{n} w_i K_H(x - y_i)
\]

where \( w \) is a vector of weights such that all \( w_i \geq 0 \) and \( \sum_i w_i = 1 \) (by default uniform \( 1/n \) weights are used), \( K_H \) is kernel \( K \) parametrized by bandwidth matrix \( H \) and \( y \) is a matrix of data points used for estimating the kernel density.

**Notice:** When using multivariate normal (Gaussian) distribution as a kernel \( K \), the bandwidth parameter \( H \) is a covariance matrix as compared to standard deviations used in univariate and product kernels.

Random generation from multivariate kernel is done by drawing with replacement rows of \( y_i \), and then adding to the sampled values random noise from multivariate normal distribution centered at the data points and parametrized by corresponding bandwidth matrix \( H \). For further details see \texttt{rmvng}.

**References**


See Also

bw.silv, density, bandwidth, kernelboot-class

Examples

set.seed(1)

# smooth bootstrap of parameters of linear regression
b1 <- kernelboot(mtcars, function(data) coef(lm(mpg ~ drat + wt, data = data)), R = 250)
b1
summary(b1)

b2 <- kernelboot(mtcars, function(data) coef(lm(mpg ~ drat + wt, data = data)), R = 250,
                  kernel = "epanechnikov")
b2
summary(b2)

# smooth bootstrap of parameters of linear regression
# smoothing phase is not applied to "am" and "cyl" variables
b3 <- kernelboot(mtcars, function(data) coef(lm(mpg ~ drat + wt + am + cyl, data = data)), R = 250,
                  ignore = c("am", "cyl"))
b3
summary(b3)

# standard bootstrap (without kernel smoothing)

b4 <- kernelboot(mtcars, function(data) coef(lm(mpg ~ drat + wt + am + cyl, data = data)), R = 250,
                  ignore = colnames(mtcars))
b4
summary(b4)

# smooth bootstrap for median of univariate data

b5 <- kernelboot(mtcars$mpg, function(data) median(data), R = 250)
b5
summary(b5)
**Description**

'kernelboot' class object

**Details**

Object of class "kernelboot", is a list with components including

- `orig.stat`: estimates from statistic on the original data,
- `boot.samples`: samples drawn,
- `call`: function call,
- `statistic`: actual statistic function that was used,
- `orig.data`: original data used for bootstrapping,
- `variables`: used variables: it is NULL for univariate data and for multivariate data it contains two lists of smoothed and ignored variables (names or column indexes) during the smoothing phase,
- `type`: type of kernel density that was used ("univariate", "product", "multivariate"),
- `param`: list of parameters that were used.

The `param` section contains:

- `R`: number of bootstrap iterations,
- `bw`: the bandwidth that was used,
- `weights`: vector of the weights that were applied,
- `kernel`: name of the kernel that was used ("multivariate", "gaussian", "epanechnikov", "rectangular", "triangular", "biweight", "cosine", "optcosine", "none"),
- `shrinked`: value of the `shrinked` parameter,
- `parallel`: indicates if parallel computation was used,
- `random.seed`: random seed used to initialize the random number generator (see `.Random.seed`).

**See Also**

`kernelboot`

---

**rmvg**

*Random generation from multivariate Gaussian kernel density*

**Description**

Random generation from multivariate Gaussian kernel density

**Usage**

`rmvg(n, y, bw = bw.silv(y), weights = NULL, adjust = 1)`
Arguments

n  
number of observations. If length(n) > 1, the length is taken to be the number required.

y  
numeric matrix or data.frame.

bw  
numeric matrix with number of rows and columns equal to ncol(y); the smoothing bandwidth to be used. This is the covariance matrix of the smoothing kernel. If provided as a single value, the same bandwidth is used for each variable. If provided as a single value, or as a vector, variables are considered as uncorrelated.

weights  
numeric vector of length equal to nrow(y); must be non-negative.

adjust  
scalar; the bandwidth used is actually adjust*bw. This makes it easy to specify values like 'half the default' bandwidth.

Details

Multivariate kernel density estimator with multivariate Gaussian (normal) kernels \( K_H \) is defined as

\[
\hat{f}_H(x) = \sum_{i=1}^{n} w_i K_H(x - y_i)
\]

where \( w \) is a vector of weights such that all \( w_i \geq 0 \) and \( \sum_i w_i = 1 \) (by default uniform \( 1/n \) weights are used), \( K_H \) is kernel \( K \) parametrized by bandwidth matrix \( H \) and \( y \) is a matrix of data points used for estimating the kernel density.

Random generation from multivariate normal distribution is possible by taking

\[
x = A'z + \mu
\]

where \( z \) is a vector of \( m \) i.i.d. standard normal deviates, \( \mu \) is a vector of means and \( A \) is a \( m \times m \) matrix such that \( A'A = \Sigma \) (\( A \) is a Cholesky factor of \( \Sigma \)). In the case of multivariate Gaussian kernel density, \( \mu \), is the \( i \)-th row of \( y \), where \( i \) is drawn randomly with replacement with probability proportional to \( w_i \), and \( \Sigma \) is the bandwidth matrix \( H \).

For functions estimating kernel densities please check KernSmooth, ks, or other packages reviewed by Deng and Wickham (2011).

References


See Also

kernelboot
Examples

set.seed(1)

dat <- mtcars[, c(1,3)]
bw <- bw.silv(dat)
X <- rmvg(5000, dat, bw = bw)

if (requireNamespace("ks", quietly = TRUE)) {
  pal <- colorRampPalette(c("chartreuse4", "yellow", "orange", "brown"))
col <- pal(10)[cut(ks::kde(dat, H = bw, eval.points = X)$estimate, breaks = 10)]

  plot(X, col = col, pch = 19, axes = FALSE,
       main = "Multivariate Gaussian Kernel")
  points(dat, pch = 2, col = "blue")
  axis(1); axis(2)
} else {

  plot(X, pch = 16, axes = FALSE, col = "#458B004D",
       main = "Multivariate Gaussian Kernel")
  points(dat, pch = 2, col = "red", lwd = 2)
  axis(1); axis(2)
}

rmvk

Random generation from product kernel density

Description

Random generation from product kernel density

Usage

rmvk(
  n,
  y,
  bw = sqrt(diag(bw.silv(y))),
  kernel = c("gaussian", "epanechnikov", "rectangular", "triangular", "biweight",
             "cosine", "optcosine"),
  weights = NULL,
  adjust = 1,
  shrinked = FALSE
)
Arguments

- `n`: number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `y`: numeric matrix or data.frame.
- `bw`: numeric vector of length equal to `ncol(y)`; the smoothing bandwidth to be used. The kernels are scaled such that this is the standard deviation of the smoothing kernel (see `density` for details). If provided as a single value, the same bandwidth is used for each variable.
- `kernel`: a character string giving the smoothing kernel to be used. This must partially match one of "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" or "optcosine", with default "gaussian", and may be abbreviated.
- `weights`: numeric vector of length equal to `nrow(y)`; must be non-negative.
- `adjust`: scalar; the bandwidth used is actually `adjust*bw`. This makes it easy to specify values like 'half the default' bandwidth.
- `shrinked`: if `TRUE` random generation algorithm preserves mean and variances of the individual variables (see `ruvk`). Shrinking is applied to each of the variables individually.

Details

Product kernel density is defined in terms of independent univariate kernels

\[ \hat{f}_H(x) = \sum_{i=1}^{n} w_i \prod_{j=1}^{m} K_{h_j}(x_i - y_{ij}) \]

where \( w \) is a vector of weights such that all \( w_i \geq 0 \) and \( \sum w_i = 1 \) (by default uniform \( 1/n \) weights are used), \( K_{h_j} \) is univariate kernel \( K \) parametrized by bandwidth \( h_j \), where \( y \) is a matrix of data points used for estimating the kernel density.

For functions estimating kernel densities please check `KernSmooth`, `ks`, or other packages reviewed by Deng and Wickham (2011).

For random generation the algorithm described in `kernelboot` is used. When using `shrinked = TRUE`, random noise is drawn from independent, shrinked univariate kernels.

References


See Also

- `kernelboot`
Examples

dat <- mtcars[, c("mpg", "disp")]

partmp <- par(mfrow = c(1, 2), mar = c(3, 3, 3, 3))

plot(rmvk(5000, dat, shrinked = FALSE), col = "#458B004D", pch = 16,
    xlim = c(0, 45), ylim = c(-200, 800),
    main = "Product kernel", axes = FALSE)
points(dat, pch = 2, lwd = 2, col = "red")
axis(1); axis(2)

plot(rmvk(5000, dat, shrinked = TRUE), col = "#458B004D", pch = 16,
    xlim = c(0, 45), ylim = c(-200, 800),
    main = "Product kernel (shrinked)", axes = FALSE)
points(dat, pch = 2, lwd = 2, col = "red")
axis(1); axis(2)

par(partmp)

cov(dat)
cov(rmvk(5000, dat, shrinked = FALSE))
cov(rmvk(5000, dat, shrinked = TRUE))

ruvk

Random generation from univariate kernel density

Description
Random generation from univariate kernel density

Usage
ruvk(
  n,
  y,
  bw = bw.nrd0(y),
  kernel = c("gaussian", "epanechnikov", "rectangular", "triangular", "biweight",
             "cosine", "optcosine"),
  weights = NULL,
  adjust = 1,
  shrinked = FALSE
)

Arguments
  n number of observations. If length(n) > 1, the length is taken to be the number required.
ruvk

y numeric vector.
bw the smoothing bandwidth to be used. The kernels are scaled such that this is the
standard deviation of the smoothing kernel (see `density` for details).
kernel a character string giving the smoothing kernel to be used. This must partially
match one of "gaussian", "rectangular", "triangular", "epanechnikov", "biweight",
"cosine" or "optcosine", with default "gaussian", and may be abbreviated.
weights numeric vector of length equal to `length(y)`; must be non-negative.
adjust scalar; the bandwidth used is actually `adjust*bw`. This makes it easy to specify
values like 'half the default' bandwidth.
shrinked if `TRUE` random generation algorithm preserves mean and variance of the original
sample.

Details

Univariate kernel density estimator is defined as

\[ f_h(x) = \frac{1}{n} \sum_{i=1}^{n} w_i K_h(x - y_i) \]

where \( w \) is a vector of weights such that all \( w_i \geq 0 \) and \( \sum_i w_i = 1 \) (by default uniform \( 1/n \)
weights are used), \( K_h = K(x/h)/h \) is kernel \( K \) parametrized by bandwidth \( h \) and \( y \) is a vector of
data points used for estimating the kernel density.

For estimating kernel densities use the `density` function.

The random generation algorithm is described in the documentation of `kernelboot` function.

References


See Also

`kernelboot, density`

Examples

```r
# ruvk() produces samples from kernel densities as estimated using
# density() function from base R
hist(ruvk(1e5, mtcars$mpg), 100, freq = FALSE, xlim = c(5, 40))
lines(density(mtcars$mpg, bw = bw.nrd0(mtcars$mpg)), col = "red")

# when using 'shrinked = TRUE', the samples differ from density() estimates
# since they are shrinked to have the same variance as the underlying data
hist(ruvk(1e5, mtcars$mpg, shrinked = TRUE), 100, freq = FALSE, xlim = c(5, 40))
lines(density(mtcars$mpg, bw = bw.nrd0(mtcars$mpg)), col = "red")
```
# Comparison of different univariate kernels under standard parametrization

kernels <- c("gaussian", "epanechnikov", "rectangular", "triangular", "biweight", "cosine", "optcosine")

partmp <- par(mfrow = c(2, 4), mar = c(3, 3, 3, 3))
for (k in kernels) {
  hist(ruvk(1e5, 0, 1, kernel = k), 25, freq = FALSE, main = k)
  lines(density(0, 1, kernel = k), col = "red")
}
par(partmp)

summary.kernelboot

## S3 method for class 'kernelboot'

summary(object, probs = c(0.025, 0.5, 0.975), ..., na.rm = FALSE)

Arguments

- **object** kernelboot class object.
- **probs** quantiles returned by `summary` (see `quantile`).
- **...** further arguments passed to or from other methods.
- **na.rm** a logical value indicating whether NA values should be stripped before the computation proceeds.
Index

RANDOM.seed, 8

bandwidth, 3, 7
bw.nrd0, 4
bw.scott (bw.silv), 2
bw.silv, 2, 4, 7
density, 4, 7, 11, 13
future_lapply, 4

kernelboot, 3, 8, 9, 11, 13
kernelboot-class, 8

multiprocess, 4

quantile, 14

rmvg, 6, 8
rmvk, 10
rnorm, 5
runif, 5
ruvk, 11, 12

summary.kernelboot, 14