Package ‘MeanShift’

August 29, 2016

Type Package

Title Clustering via the Mean Shift Algorithm

Version 1.1-1

Date 2016-02-05

Author Mattia Ciollaro and Daren Wang

Maintainer Mattia Ciollaro <mattiaciollaro@gmail.com>

Depends parallel, wavethresh

Description Clustering of vector data and functional data using the mean shift algorithm (multi-core processing is supported) or its blurring version.

License GPL-3

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2016-04-23 21:43:41

R topics documented:

MeanShift-package .................................................. 2
bmsClustering ...................................................... 2
msClustering ...................................................... 4
projectCurveWavelets ........................................... 6

Index 9
bmsClustering

MeanShift-package  Clustering via the Mean Shift Algorithm

Description

Clustering of vector data and functional data using the mean shift algorithm (multi-core processing is supported) or its blurring version.

Details

<table>
<thead>
<tr>
<th>Package:</th>
<th>MeanShift</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Title:</td>
<td>Clustering via the Mean Shift Algorithm</td>
</tr>
<tr>
<td>Version:</td>
<td>1.1-1</td>
</tr>
<tr>
<td>Date:</td>
<td>2016-02-05</td>
</tr>
<tr>
<td>Author:</td>
<td>Mattia Ciollaro and Daren Wang</td>
</tr>
<tr>
<td>Maintainer:</td>
<td>Mattia Ciollaro <a href="mailto:mattiaciollaro@gmail.com">mattiaciollaro@gmail.com</a></td>
</tr>
<tr>
<td>Depends:</td>
<td>parallel, wavethresh</td>
</tr>
<tr>
<td>Description:</td>
<td>Clustering of vector data and functional data using the mean shift algorithm (multi-core processing is supported)</td>
</tr>
<tr>
<td>License:</td>
<td>GPL-3</td>
</tr>
<tr>
<td>Suggests:</td>
<td>knitr, rmarkdown</td>
</tr>
<tr>
<td>VignetteBuilder:</td>
<td>knitr</td>
</tr>
</tbody>
</table>

See <https://normaldeviate.wordpress.com/2012/07/20/the-amazing-mean-shift-algorithm/> for a good introduction to the mean shift algorithm.

Author(s)

Mattia Ciollaro and Daren Wang
Maintainer: Mattia Ciollaro <mattiaciollaro@gmail.com>

See Also

bmsClustering msClustering projectCurveWavelets

bmsClustering  Function to perform clustering using the blurring version of the mean shift algorithm.

Description

This function implements the blurring mean shift algorithm, which approximates the standard mean shift algorithm. Because it recursively updates the entire sample at each iteration, the blurring version of the mean shift algorithm is often faster than the standard version (especially if the standard mean shift algorithm is run using a single core).
Usage

bmsClustering(X, h = NULL, kernel = "epanechnikovKernel",
tol.stop = 1e-06, max.iter = 100, tol.epsilon = 0.001)

Arguments

X
a \( p \times n \) matrix containing \( n \geq 1 \) \( p \)-dimensional numeric vectors stored as columns. Each column of \( X \) represents a sample point.

h
a strictly positive bandwidth parameter.

kernel
a kernel function (as a character string). The following kernels are supported:

- Epanechnikov: \( K(x) = \frac{3}{2}(1-x^2)I_{[0,1]}(x) \); kernel="epanechnikovKernel"
- cubic: \( K(x) = 4(1-x)^3I_{[0,1]}(x) \); kernel="cubicKernel"
- Gaussian: \( K(x) = \sqrt{\frac{2}{\pi}}e^{-x^2/2}I_{[0,\infty)}(x) \); kernel="gaussianKernel"
- exponential \( K(x) = e^{-x}I_{[0,\infty)}(x) \); kernel="exponentialKernel".

The use of the Epanechnikov kernel is recommended when using the blurring version of the mean shift algorithm.

tol.stop
a strictly positive tolerance parameter. The mean shift algorithm stops when its update generates a step of length smaller than tol.stop. tol.stop should be considerably smaller than tol.epsilon.

max.iter
a strictly positive integer specifying the maximum number of iterations before the algorithm is forced to stop.

tol.epsilon
a strictly positive tolerance parameter. Points that are less than tol.epsilon-separated are grouped in the same cluster once the algorithm stops.

Details

It is generally recommended to standardize \( X \) so that each variable has unit variance prior to running the algorithm on the data.

Roughly speaking, larger values of \( h \) produce a coarser clustering (i.e. few and large clusters). For sufficiently large values of \( h \), the algorithm produces a unique cluster containing all the data points. Smaller values of \( h \) produce a finer clustering (i.e. many small clusters). For sufficiently small values of \( h \), each cluster that is identified by the algorithm will contain exactly one data point.

If \( h \) is not specified in the function call, then \( h \) is by default set to the 30th percentile of the empirical distribution of distances between the columns of \( X \), i.e. \( h = \text{quantile}( \text{dist( t( X ) )} , 0.3 ) \).

In their implementation, gaussianKernel and exponentialKernel are rescaled to assign probability of at least 0.99 to the unit interval \([0, 1]\). This ensures that all the kernels are roughly on the same scale.

When using the blurring version of the mean shift algorithm, it is generally recommended to use a compactly supported kernel. In particular, the algorithm is guaranteed to converge in finitely many iterations with the Epanechnikov kernel.
msClustering

Function to perform clustering using the mean shift algorithm.

Description

This function implements the mean shift algorithm. The algorithm locates the modes of a kernel density estimator and associates each data point to exactly one of the modes, thus effectively clustering the data.
Usage

msClustering(X, h = NULL, kernel = "epanechnikovKernel",
tol.stop = 1e-06, tol.epsilon = 0.001, multi.core = FALSE)

Arguments

X               a \( p \times n \) matrix containing \( n \geq 1 \) \( p \)-dimensional numeric vectors stored as columns. Each column of \( X \) represents a sample point.

h               a strictly positive bandwidth parameter.

kernel         a kernel function (as a character string). The following kernels are supported:

- Epanechnikov: \( K(x) = \frac{3}{4}(1-x^2)I_{[0,1]}(x) \); kernel="epanechnikovKernel"
- cubic: \( K(x) = 4(1-x)^3I_{[0,1]}(x) \); kernel="cubicKernel"
- Gaussian: \( K(x) = \sqrt{\frac{2}{\pi}}e^{-\frac{x^2}{2}}I_{[0,\infty]}(x) \); kernel="gaussianKernel"
- exponential \( K(x) = e^{-x}I_{[0,\infty]}(x) \); kernel="exponentialKernel".

tol.stop       a strictly positive tolerance parameter. The algorithm stops when all of the updates generate steps of length smaller than tol.stop. tol.stop should be considerably smaller than tol.epsilon.

tol.epsilon    a strictly positive tolerance parameter. Points that are less than tol.epsilon-separated are grouped in the same cluster once the algorithm stops.

multi.core     logical. If TRUE, the mean shift algorithm is parallelized.

Details

It is generally recommended to standardize \( X \) so that each variable has unit variance prior to running the algorithm on the data.

Roughly speaking, larger values of \( h \) produce a coarser clustering (i.e. few and large clusters). For sufficiently large values of \( h \), the algorithm produces a unique cluster containing all the data points. Smaller values of \( h \) produce a finer clustering (i.e. many small clusters). For sufficiently small values of \( h \), each cluster that is identified by the algorithm will contain exactly one data point.

If \( h \) is not specified in the function call, then \( h \) is by default set to the 30th percentile of the empirical distribution of distances between the columns of \( X \), i.e. \( h=quantile(\ dist(\ t(\ X ) ), \ 0.3 \ ) \).

In their implementation, gaussianKernel and exponentialKernel are rescaled to assign probability of at least 0.99 to the unit interval \([0,1]\). This ensures that all the kernels are roughly on the same scale.

To specify the number of cores when multi.core=TRUE, the option mc.cores needs to be set with options( mc.cores=n.cores ), where n.cores is the number of cores that the mean shift algorithm is allowed to use for parallel computation.

Value

The function invisibly returns a list with names

components     a matrix containing the modes/cluster representatives by column.
labels         an integer vector of cluster labels.
**projectCurveWavelets**  
*Function to project a curve on a wavelet basis.*

**Description**

This function performs the Discrete Wavelet Transform (DWT) on a numeric vector representing a curve (i.e. a "functional" datum) observed on a grid and thresholds the wavelet coefficients, thus yielding a denoised and compressed representation of the same curve.

**Author(s)**

Mattia Ciollaro and Daren Wang

**References**


**See Also**

`bmsClustering`

**Examples**

```r
### an example using the iris dataset
### help( iris )

### prepare data matrix (a subset of the iris dataset)
set.seed( 2 )
indices <- sample( 1:nrow( iris ), 80 )
iris.data <- t( iris[indices,c( "Sepal.Length", "Sepal.Width" )] )

### run mean shift algorithm
clustering <- msClustering( iris.data, h=0.8 )
print( clustering )

### plot the clusters
### Not run:
plot( iris.data[1,], iris.data[2,], col=clustering$labels+2, cex=0.8, pch=16, xlab="Sepal.Length", ylab="Sepal.Width" )
points( clustering$components[1,], clustering$components[2,], col=2+( 1:ncol( clustering$components ) ), cex=1.8, pch=16 )
### End(Not run)

### using multiple cores (2)
### Not run:
options( mc.cores=2 )
clustering.mc <- msClustering( iris.data, multi.core=TRUE )
### End(Not run)
```
projectCurveWavelets

Usage

projectCurveWavelets( x, y, irreg.grid=FALSE, grid.length=NULL,
filter.number=10, family="DaubeAsymm", bc="periodic", verbose=FALSE, ... )

Arguments

x                a numeric vector of x coordinates at which the curve is observed.
y                a numeric vector of y coordinates representing the curve. x and y must have the
                same length.
irreg.grid          logical. TRUE if x is not an equispaced grid.
grid.length        a positive power of 2 or NULL (default). In order to apply the DWT, length(x)
                must be a positive power of 2. By default, if grid.length=NULL and length(x)
                is not a power of 2, x is extended to an equispaced grid whose length is positive
                power of 2 and y is extended interpolated on the extended grid. If projectCurveWavelets
                is used on multiple curves, grid.length should be set manually to ensure that
                all the discretized curves have the same length before the DWT is applied on
                each of them.
filter.number      an integer specifying the smoothness of the wavelet used in the wavelet decom-
                position of y. See the functions wd and irregwd of wavethresh for details.
family             a character string specifying the family of wavelets used in the wavelet decom-
                position of y. See the functions wd and irregwd of wavethresh for details.
bc                a character string specifying how to handle the boundary condition. See the
                functions wd and irregwd of wavethresh for details.
verbose            logical. Controls the printing of "informative" messages whilst the computation
                progresses. Such messages are generally annoying so it is turned off by default.
...                further arguments to control the thresholding of the wavelet coefficients. See
threshold.wd and threshold.irregwd of the wavethresh package for details.
                By default, projectCurveWavelets uses the default values of threshold.wd
                and threshold.irregwd to perform the thresholding of the wavelet coefficients.

Details

The function normalizes the input grid to the standard unit interval, i.e. the minimum and the
maximum values of x.grid are respectively 0 and 1.

projectCurveWavelet is designed to be used as a preliminary step towards functional clustering
using the mean shift algorithm. Given a sample of curves, projectCurveWavelet can be used to
represent each curve as a sparse vector of coefficients. These coefficients can be fed as a matrix to
msClustering or bmsClustering and clustered via the mean shift algorithm or the blurring mean
shift algorithm.

Value

The function outputs a list with names

coefficients    a numeric vector of thresholded wavelet coefficients.
projectCurveWavelets

y.wdT an object of class wd or irregwd. See `threshold.wd` and `threshold.irregwd` of the **wavethresh** package for details.
y.wavelet a numeric vector with the reconstruction of \( y \) after the application of the DWT and the thresholding of the wavelet coefficients.
x.grid the extended and equispaced grid of x values associated to y.wavelet.

Author(s)
Mattia Ciollaro and Daren Wang

References

See Also
`wavethresh` `wd` `irregwd` `threshold.wd` `threshold.irregwd` `wr` `msclustering` `bmsClustering`

Examples

```r
## generate a noisy curve observed on a regular grid
set.seed( 1 )
n.grid <- 1000
x <- seq( 2, 8, length=n.grid )
sigma.epsilon1 <- 2
sigma.epsilon2 <- 2.5
sigma.epsilon3 <- 3
sigma.epsilon4 <- 1
epsilon <- rnorm( 1000, sd=rep( c( sigma.epsilon1, sigma.epsilon2, sigma.epsilon3, sigma.epsilon4 ), rep( 250, 4 ) )
y <- x*sin( 3*x ) + 0.3*x^2 + epsilon

## project on wavelet basis with soft universal thresholding
## of the wavelet coefficients
wave <- projectCurveWavelets( x, y, type="soft", policy="universal" )

## plot wavelet reconstruction of the curve
## Not run:
x.norm <- ( x - min( x ) ) / ( max( x ) - min( x ) )
plot( x.norm, y )
lines( wave$x.grid, wave$y.wavelet, col=2, lwd=3 )
## End(Not run)

## inspect wavelet coefficients
wave.coefs <- wave$coefficients
print( length( wave.coefs ) ) ## 1023 coefficients
print( sum( wave.coefs != 0 ) ) ## only 12 are non-zero
```
Index

bmsClustering, 2, 2, 6–8
irregwd, 7, 8
MeanShift (MeanShift-package), 2
MeanShift-package, 2
msClustering, 2, 4, 4, 7, 8
projectCurveWavelets, 2, 6
threshold.irregwd, 7, 8
threshold.wd, 7, 8
wavethresh, 8
wd, 7, 8
wr, 8