Package 'curveDepth'

February 19, 2019

Version 0.1.0.9

Date 2019-02-19

Title Tukey Curve Depth and Distance in the Space of Curves

Depends Rcpp, ddalpha

Description Data recorded as paths or trajectories may be suitably described by curves, which are independent of their parametrization. For the space of such curves, the package provides functionalities for reading curves, sampling points on curves, calculating distance between curves and for computing Tukey curve depth of a curve w.r.t. to a bundle of curves. For details see Lafaye De Micheaux, Mozharovskyi, and Vimond (2019) <arXiv:1901.00180>.

LinkingTo Rcpp, RcppArmadillo

License GPL (>= 2)

LazyData TRUE

SystemRequirements C++11

RoxygenNote 6.0.1

NeedsCompilation yes

Author Pavlo Mozharovskyi [aut, cre],
Pierre Lafaye De Micheaux [aut],
Myriam Vimond [aut]

Maintainer Pavlo Mozharovskyi <pavlo.mozharovskyi@telecom-paristech.fr>

Repository CRAN

Date/Publication 2019-02-19 16:30:03 UTC

R topics documented:

`depth.curve.Tukey` *Calculate Tukey curve depth using given points*

Description

Calculates Tukey curve depth of each curve in `objects` w.r.t. the sample of curves in `data`. Calculation of partial depth of each single point can be either exact or approximate. If exact, modified method of Dyckerhoff and Mozharovskyi (2016) is used; if approximate, approximation is performed by projections on directions - points uniformly distributed on the unit hypersphere.

Usage

```
depth.curve.Tukey(objects, data, nDirs = 100L, subs = TRUE, fracInt = 0.5,
  fracEst = 0.5, subsamples = NULL, exactEst = TRUE, minMassObj = 0,
  minMassDat = 0)
```

Arguments

`objects`	A list where each element is a multivariate curve being a list containing a matrix `coords` (values, d columns).
`data`	A list where each element is a multivariate curve being a list containing a matrix `coords` (values, d columns). The depths are computed w.r.t. this data set.
`nDirs`	Number of directions used to inspect the space, drawn from the uniform distribution on the sphere.
`subs`	Whether to split each object into two disjunctive subsets (one for integrating and one for estimation) when computing the depth.
`fracInt`	Portion of an object used for integrating.
`fracEst`	Portion of an object used for estimation, maximum: `1 - fracInt`.
`subsamples`	A list indicating subsamples of points for each curve in `objects`. Each elemnt of the list corresponds to a single curve and should be given as a vector of the length equal to the number of points on it, with entries indicating: • 0 do not take the point into account at all, • 1 use point as a reference (i.e. for integrating) and thus calculate its depth, • 2 utilize point in depth calculation (i.e. for estimation).
`exactEst`	Is calculation of depth for each reference point of the curve exact (`TRUE`, by default) or approximate (`FALSE`).
`minMassObj`	Minimal portion of the `objects` distribution in the halfspace to be considered when calculating depth.
`minMassDat`	minimal portion of the `data` distribution in the halfspace to be considered when calculating depth.

Value

A vector of doubles having the same length as `objects`, whose each entry is the depth of each element of `objects` w.r.t. `data`.

References

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

Dyckerhoff, R. and Mozharovskyi P. (2016). Exact computation of the halfspace depth. *Computational Statistics and Data Analysis*, 98, 19-30.

Examples

```
library(curveDepth)
# Load digits and transform them to curves
data("mnistShort017")
n <- 10 # cardinality of each class
m <- 50 # number of points to sample
cst <- 1/10 # a threshold constant
alp <- 1/8 # a threshold constant
curves0 <- images2curves(mnistShort017$`0`[, , 1:n])
curves1 <- images2curves(mnistShort017$`1`[, , 1:n])
set.seed(1)
curves0Smpl <- sample.curves(curves0, 2 * m)
curves1Smpl <- sample.curves(curves1, 2 * m)
# Calculate depths
depthSpace = matrix(NA, nrow = n * 2, ncol = 2)
depthSpace[, 1] = depth.curve.Tukey(
  c(curves0Smpl, curves1Smpl), curves0Smpl,
  exactEst = TRUE, minMassObj = cst/m^alp)
depthSpace[, 2] = depth.curve.Tukey(
  c(curves0Smpl, curves1Smpl), curves1Smpl,
  exactEst = TRUE, minMassObj = cst/m^alp)
# Draw the DD-plot
plot(NULL, xlim = c(0, 1), ylim = c(0, 1),
     xlab = paste("Depth w.r.t. '0'"),
     ylab = paste("Depth w.r.t. '1'"),
     main = paste("DD-plot for '0' vs '1'"))
grid()
# Draw the separating rule
dat1 <- data.frame(cbind(
  depthSpace, c(rep(0, n), rep(1, n))))
ddalpha1 <- ddalpha.train(X3 ~ X1 + X2, data = dat1,
                          depth = "ddplot",
                          separator = "alpha")
ddnormal <- ddalpha1$classifiers[[1]]$hyperplane[2:3]
pts <- matrix(c(0, 0, 1, ddnormal[1] / -ddnormal[2]),
              nrow = 2, byrow = TRUE)
lines(pts, lwd = 2)
# Draw the points
points(depthSpace[1:n, ],
```

```
       col = "red", lwd = 2, pch = 1)
points(depthSpace[(n + 1):(2 * n), ],
       col = "blue", lwd = 2, pch = 3)
```

`depthc.Tukey` *Calculate Tukey curve depth for curves*

Description

Calculates Tukey curve depth of each curve in `objects` w.r.t. the sample of curves in `data`. First, `m` points are sampled from a uniform distribution on a piecewise linear approximation of each of the curves in `data` and `m / fracEst * (fracInt + fracEst)` points on each of the curves in `objects`. Second, these samples are used to calculate the Tukey curve depth.

Usage

```
depthc.Tukey(objects, data, nDirs = 100L, subs = TRUE, m = 500L,
  fracInt = 0.5, fracEst = 0.5, exactEst = TRUE, minMassObj = 0,
  minMassDat = 0)
```

Arguments

`objects`	A list where each element is a multivariate curve being a list containing a matrix `coords` (values, d columns).
`data`	A list where each element is a multivariate curve being a list containing a matrix `coords` (values, d columns). The depths are computed w.r.t. this data set.
`nDirs`	Number of directions used to inspect the space, drawn from the uniform distribution on the sphere.
`subs`	Whether to split each object into two disjunctive subsets (one for integrating and one for estimation) when computing the depth.
`m`	Number of points used for estimation.
`fracInt`	Portion of an object used for integrating.
`fracEst`	Portion of an object used for estimation, maximum: `1 - fracInt`.
`exactEst`	Is calculation of depth for each reference point of the curve exact (`TRUE`, by default) or approximate (`FALSE`).
`minMassObj`	Minimal portion of the `objects` distribution in the halfspace to be considered when calculating depth.
`minMassDat`	minimal portion of the `data` distribution in the halfspace to be considered when calculating depth.

Details

Calculation of partial depth of each single point can be either exact or approximate. If exact, an extension of the method of Dyckerhoff and Mozharovskyi (2016) is used; if approximate, approximation is performed by projections on directions - points uniformly distributed on the unit hypersphere.

Value

A vector of doubles having the same length as `objects`, whose each entry is the depth of each element of `objects` w.r.t. `data`.

References

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

Dyckerhoff, R. and Mozharovskyi P. (2016). Exact computation of the halfspace depth. *Computational Statistics and Data Analysis*, 98, 19-30.

Examples

```
library(curveDepth)
# Load digits and transform them to curves
data("mnistShort017")
n <- 10 # cardinality of each class
m <- 50 # number of points to sample
cst <- 1/10 # a threshold constant
alp <- 1/8 # a threshold constant
curves0 <- images2curves(mnistShort017$`0`[, , 1:n])
curves1 <- images2curves(mnistShort017$`1`[, , 1:n])
# Calculate depths
depthSpace = matrix(NA, nrow = n * 2, ncol = 2)
set.seed(1)
depthSpace[, 1] = depthc.Tukey(
  c(curves0, curves1), curves0, m = m,
  exactEst = TRUE, minMassObj = cst/m^alp)
depthSpace[, 2] = depthc.Tukey(
  c(curves0, curves1), curves1, m = m,
  exactEst = TRUE, minMassObj = cst/m^alp)
# Draw the DD-plot
plot(NULL, xlim = c(0, 1), ylim = c(0, 1),
     xlab = paste("Depth w.r.t. '0'"),
     ylab = paste("Depth w.r.t. '1'"),
     main = paste("DD-plot for '0' vs '1'"))
grid()
# Draw the separating rule
dat1 <- data.frame(cbind(
  depthSpace, c(rep(0, n), rep(1, n))))
ddalpha1 <- ddalpha.train(X3 ~ X1 + X2, data = dat1,
                          depth = "ddplot",
                          separator = "alpha")
ddnormal <- ddalpha1$classifiers[[1]]$hyperplane[2:3]
pts <- matrix(c(0, 0, 1, ddnormal[1] / -ddnormal[2]),
              nrow = 2, byrow = TRUE)
lines(pts, lwd = 2)
# Draw the points
points(depthSpace[1:n, ],
       col = "red", lwd = 2, pch = 1)
points(depthSpace[(n + 1):(2 * n), ],
```

```
        col = "blue", lwd = 2, pch = 3)
```

`dist.curves` *Distance for curves*

Description

Calculates distance matrix for a sample of curves using the minimax metric.

Usage

```
dist.curves(curves, oneWay = FALSE, verbosity = 0L)
```

Arguments

`curves`	A list where each element is a function being a list containing a matrix `coords` (values, d columns).
`oneWay`	Whether curves should be condisered as a one-directional, `FALSE` by default.
`verbosity`	Level of reporting messages, the higher the more progress reports are printed, set `0` (default) for no messages.

Value

A matrix `length(curves) x length(curves)` with each entry being the distance between two curves.

References

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

Examples

```
library(curveDepth)
# Pixel-grid filling function for an image
plotGridImage <- function(dims){
  redDims1 <- dims[1] - 1
  redDims2 <- dims[2] - 1
  for (i in 1:(dims[1] - 1)){
    lines(c(i / redDims1 - 0.5 / redDims1,
            i / redDims1 - 0.5 / redDims1),
          c(0 - 0.5 / redDims2, 1 + 0.5 / redDims2),
          lwd = 1, lty = 3, col = "lightgray")
    lines(c(0 - 0.5 / redDims1, 1 + 0.5 / redDims1),
          c(i / redDims2 - 0.5 / redDims2,
            i / redDims2 - 0.5 / redDims2),
          lwd = 1, lty = 3, col = "lightgray")
  }
```

```
  rect(0 - 0.5 / redDims1, 0 - 0.5 / redDims2,
       1 + 0.5 / redDims1, 1 + 0.5 / redDims2)
}
# Load two Sevens and one One, plot them,
# and transform to curves
data("mnistShort017")
# First Seven
firstSevenDigit <- mnistShort017$`7`[, , 5]
image(as.matrix(rev(as.data.frame(firstSevenDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(firstSevenDigit)[1:2])
firstSevenCurve <- images2curves(array(
  firstSevenDigit, dim = c(28, 28, 1)))[[1]]
# Second Seven
secondSevenDigit <- mnistShort017$`7`[, , 6]
image(as.matrix(rev(as.data.frame(secondSevenDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(secondSevenDigit)[1:2])
secondSevenCurve <- images2curves(array(
  secondSevenDigit, dim = c(28, 28, 1)))[[1]]
# A One
aOneDigit <- mnistShort017$`1`[, , 1]
image(as.matrix(rev(as.data.frame(aOneDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(aOneDigit)[1:2])
aOneCurve <- images2curves(array(
  aOneDigit, dim = c(28, 28, 1)))[[1]]
# Caculate distances between all the curves
distMatrix <- dist.curves(list(
  firstSevenCurve, secondSevenCurve, aOneCurve))
# Print distance matrix
print(distMatrix)
```

`dist.curves.asymm` *Distance for curves*

Description

Calculates distance matrix for two samples of curves using minimax metric. The function can be particularly useful for parallel computation of a big distance matrix.

Usage

```
dist.curves.asymm(curvesRows, curvesCols, oneWay = FALSE, verbosity = 0L)
```

Arguments

curvesRows	A list where each element is a function being a list containing a matrix coords (values, d columns).
curvesCols	A list where each element is a function being a list containing a matrix coords (values, d columns).
oneWay	Whether curves should be condisered as a one-directional, FALSE by default.
verbosity	Level of reporting messages, the higher the more progress reports are printed, set 0 (default) for no messages.

Value

A matrix length(curvesRows) x length(curvesCols) with each entry being the distance between two corresponding curves.

References

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

Examples

```
library(curveDepth)
# Pixel-grid filling function for an image
plotGridImage <- function(dims){
  redDims1 <- dims[1] - 1
  redDims2 <- dims[2] - 1
  for (i in 1:(dims[1] - 1)){
    lines(c(i / redDims1 - 0.5 / redDims1,
            i / redDims1 - 0.5 / redDims1),
          c(0 - 0.5 / redDims2, 1 + 0.5 / redDims2),
          lwd = 1, lty = 3, col = "lightgray")
    lines(c(0 - 0.5 / redDims1, 1 + 0.5 / redDims1),
          c(i / redDims2 - 0.5 / redDims2,
            i / redDims2 - 0.5 / redDims2),
          lwd = 1, lty = 3, col = "lightgray")
  }
  rect(0 - 0.5 / redDims1, 0 - 0.5 / redDims2,
       1 + 0.5 / redDims1, 1 + 0.5 / redDims2)
}
# Load two Sevens and one One, plot them,
# and transform to curves
data("mnistShort017")
# First Seven
firstSevenDigit <- mnistShort017$`7`[, , 5]
image(as.matrix(rev(as.data.frame(firstSevenDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(firstSevenDigit)[1:2])
firstSevenCurve <- images2curves(array(
```

```
  firstSevenDigit, dim = c(28, 28, 1)))[[1]]
# Second Seven
secondSevenDigit <- mnistShort017$`7`[, , 6]
image(as.matrix(rev(as.data.frame(secondSevenDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(secondSevenDigit)[1:2])
secondSevenCurve <- images2curves(array(
  secondSevenDigit, dim = c(28, 28, 1)))[[1]]
# A One
aOneDigit <- mnistShort017$`1`[, , 1]
image(as.matrix(rev(as.data.frame(aOneDigit))),
  col = gray((255:0) / 256), asp = 1,
  xlim = c(0 - 1 / 27, 1 + 1 / 27),
  ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(aOneDigit)[1:2])
aOneCurve <- images2curves(array(
  aOneDigit, dim = c(28, 28, 1)))[[1]]
# Caculate distances between all the curves
distMatrix <- matrix(0, 3, 3)
distMatrix[3, 1:2] <- distMatrix[1:2, 3] <-
  dist.curves.asymm(list(
    firstSevenCurve, secondSevenCurve), list(aOneCurve))
distMatrix[2, 1] <- distMatrix[1, 2] <-
  dist.curves.asymm(
    list(firstSevenCurve), list(secondSevenCurve))
# Print distance matrix
print(distMatrix)
```

`dist.images` *Distance for images*

Description

Calculates distance matrix for a sample of images using the minimax metric. This fucntion can be seen as a wrapper of a sequential call of `images2curves` and `dist.curves`.

Usage

```
dist.images(images, verbosity = 0L)
```

Arguments

`images`	A 3-dimensional array with each slice (matrix in first two dimensions) corresponding to an image. Each (eps-strictly) positive entry is regarded as an occupied pixel (one), otherwise it is regarded as an empty pixel, of an image.
`verbosity`	Level of reporting messages, the higher the more progress reports are printed, set `0` (default) for no messages.

Value

A matrix `dim(images)[3] x dim(images)[3]` with each entry being the distance between two images.

References

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

Examples

```
library(curveDepth)
# Pixel-grid filling function for an image
plotGridImage <- function(dims){
  redDims1 <- dims[1] - 1
  redDims2 <- dims[2] - 1
  for (i in 1:(dims[1] - 1)){
    lines(c(i / redDims1 - 0.5 / redDims1,
            i / redDims1 - 0.5 / redDims1),
          c(0 - 0.5 / redDims2, 1 + 0.5 / redDims2),
          lwd = 1, lty = 3, col = "lightgray")
    lines(c(0 - 0.5 / redDims1, 1 + 0.5 / redDims1),
          c(i / redDims2 - 0.5 / redDims2,
            i / redDims2 - 0.5 / redDims2),
          lwd = 1, lty = 3, col = "lightgray")
  }
  rect(0 - 0.5 / redDims1, 0 - 0.5 / redDims2,
       1 + 0.5 / redDims1, 1 + 0.5 / redDims2)
}
# Load two Sevens and one One, and plot them
data("mnistShort017")
# First Seven
  firstSevenDigit <- mnistShort017$`7`[, , 5]
image(as.matrix(rev(as.data.frame(firstSevenDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(firstSevenDigit)[1:2])
# Second Seven
secondSevenDigit <- mnistShort017$`7`[, , 6]
image(as.matrix(rev(as.data.frame(secondSevenDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(secondSevenDigit)[1:2])
# A One
aOneDigit <- mnistShort017$`1`[, , 1]
image(as.matrix(rev(as.data.frame(aOneDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(aOneDigit)[1:2])
```

```
# Caculate distances between all the images
threeDigits <- array(NA, dim = c(nrow(firstSevenDigit),
  ncol(firstSevenDigit), 3))
threeDigits[, , 1] <- firstSevenDigit
threeDigits[, , 2] <- secondSevenDigit
threeDigits[, , 3] <- aOneDigit
distMatrix <- dist.images(threeDigits)
# Print distance matrix
print(distMatrix)
```

`images2curves` *Convert images to curves*

Description

Converts images to curves with points sorted in traversing order.

Usage

```
images2curves(images)
```

Arguments

`images`	A 3-dimensional array with each slice (matrix in first two dimensions) corresponding to an image. Each (eps-strictly) positive entry is regarded as an occupied pixel (one), otherwise it is regarded as an empty pixel, of an image.

Value

A list of curves where each element is a function being a list containing a matrix `coords` (curve's values, d columns).

References

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

Examples

```
library(curveDepth)
# Pixel-grid filling function for an image
plotGridImage <- function(dims){
  redDims1 <- dims[1] - 1
  redDims2 <- dims[2] - 1
  for (i in 1:(dims[1] - 1)){
    lines(c(i / redDims1 - 0.5 / redDims1,
            i / redDims1 - 0.5 / redDims1),
            c(0 - 0.5 / redDims2, 1 + 0.5 / redDims2),
            lwd = 1, lty = 3, col = "lightgray")
```

```
    lines(c(0 - 0.5 / redDims1, 1 + 0.5 / redDims1),
          c(i / redDims2 - 0.5 / redDims2,
            i / redDims2 - 0.5 / redDims2),
            lwd = 1, lty = 3, col = "lightgray")
  }
  rect(0 - 0.5 / redDims1, 0 - 0.5 / redDims2,
       1 + 0.5 / redDims1, 1 + 0.5 / redDims2)
}
# Pixel-grid filling function for a curve
plotGridCurve <- function(dims){
  for (i in 1:(dims[1] - 1)){
    lines(c(i / dims[1], i / dims[1]), c(0, 1),
          lwd = 1, lty = 3, col = "lightgray")
    lines(c(0, 1), c(i / dims[2], i / dims[2]),
          lwd = 1, lty = 3, col = "lightgray")
  }
  rect(0, 0, 1, 1)
}
# Load a digit and plot it
data("mnistShort017")
  aSevenDigit <- mnistShort017$`7`[, , 5]
image(as.matrix(rev(as.data.frame(aSevenDigit))),
      col = gray((255:0) / 256), asp = 1,
      xlim = c(0 - 1 / 27, 1 + 1 / 27),
      ylim = c(0 - 1 / 27, 1 + 1 / 27))
plotGridImage(dim(aSevenDigit)[1:2])
# Convert the digit to a curve and plot it
aSevenCurve <- images2curves(array(
  aSevenDigit, dim = c(28, 28, 1)))[[1]]
plot(cbind(aSevenCurve$coords[, 1],
           1 - aSevenCurve$coords[, 2]),
           type = "l", lwd = 3, asp = 1,
           xlim = c(0, 1), ylim = c(0, 1),
           xlab = "x", ylab = "y")
  plotGridCurve(dim(aSevenDigit)[1:2])
```

`mnistShort017` *A short version of the MNIST data set*

Description

First 100 digits from the MNIST data set belonging to classes '0', '1', and '7' representable as curves agter skeletonization.

Usage

```
data("mnistShort017")
```

Format

A list of three elements '0', '1', and '7', each being an array with dimensions `(28, 28, 100)`. Each slice of each of these arrays, in third dimension, contains a single monochrome digit image represented by a `28x28` indicator matrix. See Lafaye De Micheaux, Mozharovksyi and Vimond (2018) and accompanying codes for details on preprocessing.

Author(s)

Yann LeCun (Courant Institute, NYU), Corinna Cortes (Google Labs, New York), Christopher J.C. Burges (Microsoft Research, Redmond),

preprocessing performed by Myriam Vimond (CREST, Ensai, University of Bretagne Loire).

Source

http://yann.lecun.com/exdb/mnist/

References

LeCun, Y., Bottou, L.,Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

`sample.curves` *Sample points on curves*

Description

Samples points uniformly on curves interpolated as linear consequent segments.

Usage

```
sample.curves(curves, ptsPerCurve = as.integer(c(500)))
```

Arguments

`curves`	A list where each element is a function being a list containing a matrix `coords` (curves' values, d columns).
`ptsPerCurve`	A vector of numbers of points to be sampled on each curve. If `length(ptsPerCurve) < length(curves)` then the first entry of `ptsPerCurve` is considered only, and corresponds to the number of points on a curve.

Value

A list of curves with each entry being a list constiting of [[1]] the drawn curve being a matrix named `coords`, [[2]] length of the curve as in `curves` named `length.init`, and [[3]] length of the drawn curve named `length`.

References

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

Examples

```
library(curveDepth)
# Load digits and transform them to curves
data("mnistShort017")
n <- 10 # cardinality of each class
m <- 50 # number of points to sample
cst <- 1/10 # a threshold constant
alp <- 1/8 # a threshold constant
curves0 <- images2curves(mnistShort017$`0`[, , 1:n])
curves1 <- images2curves(mnistShort017$`1`[, , 1:n])
set.seed(1)
curves0Smpl <- sample.curves(curves0, 2 * m)
curves1Smpl <- sample.curves(curves1, 2 * m)
# Calculate depths
depthSpace = matrix(NA, nrow = n * 2, ncol = 2)
depthSpace[, 1] = depth.curve.Tukey(
  c(curves0Smpl, curves1Smpl), curves0Smpl,
  exactEst = TRUE, minMassObj = cst/m^alp)
depthSpace[, 2] = depth.curve.Tukey(
  c(curves0Smpl, curves1Smpl), curves1Smpl,
  exactEst = TRUE, minMassObj = cst/m^alp)
# Draw the DD-plot
plot(NULL, xlim = c(0, 1), ylim = c(0, 1),
     xlab = paste("Depth w.r.t. '0'"),
     ylab = paste("Depth w.r.t. '1'"),
     main = paste("DD-plot for '0' vs '1'"))
grid()
# Draw the separating rule
dat1 <- data.frame(cbind(
  depthSpace, c(rep(0, n), rep(1, n))))
ddalpha1 <- ddalpha.train(X3 ~ X1 + X2, data = dat1,
                          depth = "ddplot",
                          separator = "alpha")
ddnormal <- ddalpha1$classifiers[[1]]$hyperplane[2:3]
pts <- matrix(c(0, 0, 1, ddnormal[1] / -ddnormal[2]),
              nrow = 2, byrow = TRUE)
lines(pts, lwd = 2)
# Draw the points
points(depthSpace[1:n, ],
       col = "red", lwd = 2, pch = 1)
points(depthSpace[(n + 1):(2 * n), ],
       col = "blue", lwd = 2, pch = 3)
```

`voxelize` *Voxelization of functions*

Description

Convertes a pice-wise linear parametrized funtion into a discretized voxel representation.

Usage

```
voxelize(f, from, to, by)
```

Arguments

f	A parametrized function as a list containing a vector "args" (arguments), and a matrix "vals" (values, d columns).
from	A vector of d numbers, each giving a starting discretization point for one dimension.
to	A vector of d numbers, each giving a finishing discretization point for one dimension.
by	A vector of d numbers, each giving discretization step for one dimension.

Value

A list containing two matrices: "voxels" with rows being voxel numbers, and "coords" with rows being coordinates of voxel centers.

References

Lafaye De Micheaux, P., Mozharovskyi, P. and Vimond, M. (2018). Depth for curve data and applications.

Examples

```
library(curveDepth)
# Create some data based on growth curves
g1d <- dataf.growth()
g3d <- list("")
set.seed(1)
for (i in 1:length(g1d$dataf)){
  g3d[[i]] <- list(
    args = g1d$dataf[[1]]$args,
    vals = cbind(g1d$dataf[[i]]$vals,
                 g1d$dataf[[i]]$vals[length(g1d$dataf[[i]]$vals):1],
                 rnorm(length(g1d$dataf[[i]]$vals), sd = 1) +
                   rnorm(1, mean = 0, sd = 10)))
}
# Define voxels' bounds and resolution
from <- c(65, 65, -25)
to <- c(196, 196, 25)
steps <- 100
by <- (to - from) / steps
# Voxelize all curves
fs <- list("")
for (i in 1:length(g3d)){
```

```
  fs[[i]] <- voxelize(g3d[[i]], from, to, by)
}
## Not run:
# Plot first 10 curves
library(rgl)
rgl.open()
rgl.bg(color = "white")
for (i in 1:10){
  spheres3d(fs[[i]]$voxels[, 1], fs[[i]]$voxels[, 2], fs[[i]]$voxels[, 3],
            col = "red", radius = 0.5)
}
## End(Not run)
```

Index