

# Package ‘ddalpha’

March 23, 2022

**Type** Package

**Title** Depth-Based Classification and Calculation of Data Depth

**Version** 1.3.13

**Date** 2022-03-22

**SystemRequirements** C++11

**Depends** R (>= 2.10), stats, utils, graphics, grDevices, MASS, class,  
robustbase, sfsmisc, geometry

**Imports** Rcpp (>= 0.11.0)

**LinkingTo** BH, Rcpp

**Description** Contains procedures for depth-based supervised learning, which are entirely non-parametric, in particular the DDalpha-procedure (Lange, Mosler and Mozharovskyi, 2014 <[doi:10.1007/s00362-012-0488-4](https://doi.org/10.1007/s00362-012-0488-4)>). The training data sample is transformed by a statistical depth function to a compact low-dimensional space, where the final classification is done. It also offers an extension to functional data and routines for calculating certain notions of statistical depth functions. 50 multivariate and 5 functional classification problems are included. (Pokotylo, Mozharovskyi and Dyckerhoff, 2019 <[doi:10.18637/jss.v091.i05](https://doi.org/10.18637/jss.v091.i05)>).

**License** GPL-2

**NeedsCompilation** yes

**Author** Oleksii Pokotylo [aut, cre],  
Pavlo Mozharovskyi [aut],  
Rainer Dyckerhoff [aut],  
Stanislav Nagy [aut]

**Maintainer** Oleksii Pokotylo <[alexey.pokotylo@gmail.com](mailto:alexey.pokotylo@gmail.com)>

**Repository** CRAN

**Date/Publication** 2022-03-23 07:50:16 UTC

## R topics documented:

ddalpha-package	3
Cmetric	6

compclassf.classify . . . . .	7
compclassf.train . . . . .	8
Custom Methods . . . . .	10
dataf . . . . .	15
dataf.* . . . . .	17
dataf.geneexp . . . . .	18
dataf.growth . . . . .	19
dataf.medflies . . . . .	20
dataf.population . . . . .	22
dataf.population2010 . . . . .	23
dataf.sim.1.CFF07 . . . . .	25
dataf.sim.2.CFF07 . . . . .	26
dataf.tecator . . . . .	28
dataf2rawfd . . . . .	29
ddalpha.classify . . . . .	31
ddalpha.getErrorRateCV . . . . .	33
ddalpha.getErrorRatePart . . . . .	34
ddalpha.test . . . . .	35
ddalpha.train . . . . .	37
ddalphaf.classify . . . . .	43
ddalphaf.getErrorRateCV . . . . .	45
ddalphaf.getErrorRatePart . . . . .	46
ddalphaf.test . . . . .	47
ddalphaf.train . . . . .	49
depth. . . . .	51
depth.betaSkeleton . . . . .	53
depth.contours . . . . .	55
depth.contours.ddalpha . . . . .	56
depth.graph . . . . .	58
depth.halfspace . . . . .	59
depth.L2 . . . . .	61
depth.Mahalanobis . . . . .	63
depth.potential . . . . .	65
depth.projection . . . . .	67
depth.qhpeeling . . . . .	69
depth.sample . . . . .	70
depth.simplicial . . . . .	71
depth.simplicialVolume . . . . .	73
depth.space. . . . .	75
depth.space.halfspace . . . . .	76
depth.space.Mahalanobis . . . . .	78
depth.space.potential . . . . .	79
depth.space.projection . . . . .	81
depth.space.simplicial . . . . .	83
depth.space.simplicialVolume . . . . .	84
depth.space.spatial . . . . .	86
depth.space.zonoid . . . . .	87
depth.spatial . . . . .	89

depth.zonoid . . . . .	90
depthf. . . . .	92
depthf.ABD . . . . .	94
depthf.BD . . . . .	95
depthf.fd1 . . . . .	96
depthf.fd2 . . . . .	98
depthf.hM . . . . .	100
depthf.hM2 . . . . .	101
depthf.HR . . . . .	103
depthf.RP1 . . . . .	104
depthf.RP2 . . . . .	106
depthf.simplicialBand . . . . .	107
derivatives.est . . . . .	109
dknn.classify . . . . .	110
dknn.classify.trained . . . . .	111
dknn.train . . . . .	113
draw.ddplot . . . . .	114
FKS . . . . .	115
getdata . . . . .	117
infimalRank . . . . .	120
is.in.convex . . . . .	121
L2metric . . . . .	122
plot.ddalpha . . . . .	123
plot.ddalphaf . . . . .	124
plot.functional . . . . .	125
rawfd2dataf . . . . .	127
resetPar . . . . .	128
shape.fd.analysis . . . . .	129
shape.fd.outliers . . . . .	131

**Index****134**

ddalpha-package

*Depth-Based Classification and Calculation of Data Depth***Description**

The package provides many procedures for calculating the depth of points in an empirical distribution for many notions of data depth. Further it provides implementations for depth-based classification, for multivariate and functional data.

The package implements the  $DD_\alpha$ -classifier (Lange, Mosler and Mozharovskyi, 2014), a nonparametric procedure for supervised binary classification with  $q \geq 2$  classes. In the training step, the sample is first transformed into a  $q$ -dimensional cube of depth vectors, then a linear separation rule in its polynomial extension is constructed with the  $\alpha$ -procedure. The classification step involves alternative treatments of 'outsiders'.

**Details**

Package: ddalpha  
 Type: Package  
 Version: 1.3.13  
 Date: 2022-03-22  
 License: GPL-2

Use `ddalpha.train` to train the DD-classifier and `ddalpha.classify` to classify with it. Load sample classification problems using `getdata`. The package contains 50 classification problems built of 33 sets of real data.

The list of the implemented multivariate depths is found in topic `depth.`, for functional depths see `depthf.`. The depth representations of the multivariate data are obtained with `depth.space.`. Functions `depth.contours` and `depth.contours.ddalpha` build depth contours, and `depth.graph` builds depth graphs for two-dimensional data. Function `draw.ddplot` draws DD-plot for the existing DD-classifier, or for pre-calculated depth space.

The package supports user-defined depths and classifiers, see topic `Custom Methods`. A pre-calculated DD-plot may also be used as data, see topic `ddalpha.train`.

`is.in.convex` shows whether an object is no 'outsider', i.e. can be classified by its depth values. Outsiders are alternatively classified by LDA, kNN and maximum Mahalanobis depth as well as by random assignment.

Use `compclassf.train` and `ddalphaf.train` to train the functional DD-classifiers and `compclassf.classify` `ddalpha.classify` to classify with them. Load sample functional classification problems with `dataf.*`. The package contains 4 functional data sets and 2 data set generators. The functional data are visualized with `plot.functional`.

### Author(s)

Oleksii Pokotylo, <alexey.pokotylo@gmail.com>  
 Pavlo Mozharovskyi, <pavlo.mozharovskyi@telecom-paris.fr>  
 Rainer Dyckerhoff, <rainer.dyckerhoff@statistik.uni-koeln.de>  
 Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### References

- Pokotylo, O., Mozharovskyi, P., Dyckerhoff, R. (2019). Depth and depth-based classification with R-package ddalpha. *Journal of Statistical Software* **91** 1–46.
- Lange, T., Mosler, K., and Mozharovskyi, P. (2014). Fast nonparametric classification based on data depth. *Statistical Papers* **55** 49–69.
- Lange, T., Mosler, K., and Mozharovskyi, P. (2014).  $DD\alpha$ -classification of asymmetric and fat-tailed data. In: Spiliopoulou, M., Schmidt-Thieme, L., Janning, R. (eds), *Data Analysis, Machine Learning and Knowledge Discovery*, Springer (Berlin), 71–78.
- Mosler, K. and Mozharovskyi, P. (2017). Fast DD-classification of functional data. *Statistical Papers* **58** 1055–1089.
- Mozharovskyi, P. (2015). *Contributions to Depth-based Classification and Computation of the Tukey Depth*. Verlag Dr. Kovac (Hamburg).

Mozharovskyi, P., Mosler, K., and Lange, T. (2015). Classifying real-world data with the DD $\alpha$ -procedure. *Advances in Data Analysis and Classification* **9** 287–314.

Nagy, S., Gijbels, I. and Hlubinka, D. (2017). Depth-based recognition of shape outlying functions. *Journal of Computational and Graphical Statistics*. To appear.

### See Also

`ddalpha.train`, `ddalpha.classify`,  
`ddalphaf.train`, `ddalphaf.classify`, `compclassf.train`, `compclassf.classify`  
`depth.`, `depthf.`, `depth.space.`,  
`is.in.convex`,  
`getdata`, `dataf.*`,  
`plot.ddalpha`, `plot.ddalphaf`, `plot.functional`, `depth.graph`, `draw.ddplot`.

### Examples

```
# Generate a bivariate normal location-shift classification task
# containing 200 training objects and 200 to test with
class1 <- mvrnorm(200, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
trainIndices <- c(1:100)
testIndices <- c(101:200)
propertyVars <- c(1:2)
classVar <- 3
trainData <- rbind(cbind(class1[trainIndices,], rep(1, 100)),
                  cbind(class2[trainIndices,], rep(2, 100)))
testData <- rbind(cbind(class1[testIndices,], rep(1, 100)),
                  cbind(class2[testIndices,], rep(2, 100)))
data <- list(train = trainData, test = testData)

# Train the DDalpha-classifier
ddalpha <- ddalpha.train(data$train)

# Classify by means of DDalpha-classifier
classes <- ddalpha.classify(ddalpha, data$test[,propertyVars])
cat("Classification error rate:",
    sum(unlist(classes) != data$test[,classVar])/200, "\n")

# Calculate zonoid depth of top 10 testing objects w.r.t. 1st class
depths.zonoid <- depth.zonoid(data$test[1:10,propertyVars],
                              data$train[trainIndices,propertyVars])
cat("Zonoid depths:", depths.zonoid, "\n")

# Calculate the random Tukey depth of top 10 testing objects w.r.t. 1st class
depths.halfspace <- depth.halfspace(data$test[1:10,propertyVars],
                                    data$train[trainIndices,propertyVars])
cat("Random Tukey depths:", depths.halfspace, "\n")
```

```
# Calculate depth space with zonoid depth
dspace.zonoid <- depth.space.zonoid(data$train[,propertyVars], c(100, 100))

# Calculate depth space with the exact Tukey depth
dspace.halfspace <- depth.space.halfspace(data$train[,propertyVars], c(100, 100), exact = TRUE)

# Count outsiders
numOutsiders = sum(rowSums(is.in.convex(data$test[,propertyVars],
                                       data$train[,propertyVars], c(100, 100))) == 0)
cat(numOutsiders, "outsiders found in the testing sample.\n")
```

---

Cmetric

*Fast Computation of the Uniform Metric for Sets of Functional Data*


---

## Description

Returns the matrix of  $C$  (uniform) distances between two sets of functional data.

## Usage

```
Cmetric(A, B)
```

## Arguments

- |   |  |
|---|--|
| A | Functions of the first set, represented by a matrix of their functional values of size $m \times d$ . $m$ stands for the number of functions, $d$ is the number of the equidistant points in the domain of the data at which the functional values of the $m$ functions are evaluated.   |
| B | Functions of the second set, represented by a matrix of their functional values of size $n \times d$ . $n$ stands for the number of functions, $d$ is the number of the equidistant points in the domain of the data at which the functional values of the $n$ functions are evaluated. The grid of observation points for the functions A and B must be the same. |

## Details

For two sets of functional data of sizes  $m$  and  $n$  represented by matrices of their functional values, this function returns the symmetric matrix of size  $m \times n$  whose entry in the  $i$ -th row and  $j$ -th column is the approximated  $C$  (uniform) distance of the  $i$ -th function from the first set, and the  $j$ -th function from the second set. This function is utilized in the computation of the h-mode depth.

## Value

A symmetric matrix of the distances of the functions of size  $m \times n$ .

## Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**See Also**

[depthf.hm](#)  
[dataf2rawfd](#)

**Examples**

```
datapop = dataf2rawfd(dataf.population()$dataf, range=c(1950, 2015), d=66)
A = datapop[1:20,]
B = datapop[21:50,]
Cmetric(A,B)
```

---

compclassf.classify    *Classify using Functional Componentwise Classifier*

---

**Description**

Classifies data using the functional componentwise classifier.

**Usage**

```
compclassf.classify(compclassf, objectsf, subset, ...)
```

```
## S3 method for class 'compclassf'
predict(object, objectsf, subset, ...)
```

**Arguments**

compclassf, object	Functional componentwise classifier (obtained by <a href="#">compclassf.train</a> ).
objectsf	list containing lists (functions) of two vectors of equal length, named "args" and "vals": arguments sorted in ascending order and corresponding them values respectively
subset	an optional vector specifying a subset of observations to be classified.
...	additional parameters, passed to the classifier, selected with parameter classifier.type in <a href="#">compclassf.train</a> .

**Value**

List containing class labels.

**References**

Delaigle, A., Hall, P., and Bathia, N. (2012). Componentwise classification and clustering of functional data. *Biometrika* **99** 299–313.

**See Also**

[compclassf.train](#) to train the functional componentwise classifier.

**Examples**

```
## Not run:
## load the Growth dataset
dataf = dataf.growth()

learn = c(head(dataf$dataf, 49), tail(dataf$dataf, 34))
labels =c(head(dataf$labels, 49), tail(dataf$labels, 34))
test = tail(head(dataf$dataf, 59), 10) # elements 50:59. 5 girls, 5 boys

c = compclassf.train (learn, labels, classifier.type = "ddalpha")

classified = compclassf.classify(c, test)

print(unlist(classified))

## End(Not run)
```

---

compclassf.train      *Functional Componentwise Classifier*

---

**Description**

Trains the functional componentwise classifier

**Usage**

```
compclassf.train (dataf, labels, subset,
                  to.equalize = TRUE,
                  to.reduce = TRUE,
                  classifier.type = c("ddalpha", "maxdepth", "knnaff", "lda", "qda"),
                  ...)
```

**Arguments**

dataf	list containing lists (functions) of two vectors of equal length, named "args" and "vals": arguments sorted in ascending order and corresponding them values respectively
labels	list of output labels of the functional observations
subset	an optional vector specifying a subset of observations to be used in training the classifier.
to.equalize	Adjust the data to have equal (the largest) argument interval.

to.reduce        If the data spans a subspace only, project on it (by PCA).  
 classifier.type    the classifier which is used on the transformed space. The default value is 'ddalpha'.  
 ...                additional parameters, passed to the classifier, selected with parameter classifier.type.

### Details

The finite-dimensional space is directly constructed from the observed values. Delaigle, Hall and Bathia (2012) consider (almost) all sets of discretization points that have a given cardinality.

The usual classifiers are then trained on the constructed finite-dimensional space.

### Value

Trained functional componentwise classifier

### References

Delaigle, A., Hall, P., and Bathia, N. (2012). Componentwise classification and clustering of functional data. *Biometrika* **99** 299–313.

### See Also

[compclassf.classify](#) for classification using functional componentwise classifier,  
[ddalphaf.train](#) to train the functional DD-classifier,  
[dataf.\\*](#) for functional data sets included in the package.

### Examples

```
## Not run:
## load the Growth dataset
dataf = dataf.growth()

learn = c(head(dataf$dataf, 49), tail(dataf$dataf, 34))
labels = c(head(dataf$labels, 49), tail(dataf$labels, 34))
test = tail(head(dataf$dataf, 59), 10) # elements 50:59. 5 girls, 5 boys

c = compclassf.train(learn, labels, classifier.type = "ddalpha")

classified = compclassf.classify(c, test)

print(unlist(classified))

## End(Not run)
```

**Description**

To use a custom depth function or classifier one has to implement three functions: parameters validator, learning and calculating functions.

**Details****To define a depth function:**

**.<NAME>\_validate** validates parameters passed to `ddalpha.train` and passes them to the `ddalpha` object.

IN:

`ddalpha` the `ddalpha` object, containing the data and settings (mandatory)  
`<custom parameters>` parameters that are passed to the user-defined method  
`...` other parameters (mandatory)

OUT:

`list()` list of output parameters, after the validation is finished, these parameters are stored in the `ddalpha`

**.<NAME>\_learn** trains the depth

IN:

`ddalpha` the `ddalpha` object, containing the data and settings

MODIFIES:

`ddalpha` store the calculated statistics in the `ddalpha` object

`depths` calculate the depths of each pattern, e.g.

for (i in 1:ddalpha\$numPatterns) ddalpha\$patterns[[i]]\$depths = .<NAME>\_depths(ddalpha, dda

OUT:

`ddalpha` the updated `ddalpha` object

**.<NAME>\_depths** calculates the depths

IN:

`ddalpha` the `ddalpha` object, containing the data and settings

`objects` the objects for which the depths are calculated

OUT:

`depths` the calculated depths for each object (rows), with respect to each class (cols)

Usage: `ddalpha.train(data, depth = "<NAME>", <custom parameters>, ...)`

#### Custom depths ####

```
.MyDepth_validate <- function(ddalpha, mydepth.parameter = "value", ...){
  print("MyDepth validating")
  # validate the parameters
```

```

if (!is.valid(mydepth.parameter)){
  warning("Argument \"mydepth.parameter\" not specified correctly. Default value is used")
  mydepth.parameter = "value"
  # or stop("Argument \"mydepth.parameter\" not specified correctly.")
}
# the values from the return list will be stored in the ddalpha object
return (list(mydepthpar = mydepth.parameter))
}
.MyDepth_learn <- function(ddalpha){
  print("MyDepth learning")
  #1. Calculate any statistics based on data that .MyDepth_depths needs
  # and store them to the ddalpha object:
  ddalpha$mydepth.statistic = "some value"
  #2. Calculate depths for each pattern
  for (i in 1:ddalpha$numPatterns){
    ddalpha$patterns[[i]]$depths = .MyDepth_depths(ddalpha, ddalpha$patterns[[i]]$points)
  }
  return(ddalpha)
}
.MyDepth_depths <- function(ddalpha, objects){
  print("MyDepth calculating")
  depths <- NULL
  # The depth parameters are accessible in the ddalpha object:
  mydepth.parameter = ddalpha$mydepth.parameter
  mydepth.statistic = ddalpha$mydepth.statistic
  #calculate the depths of the objects w.r.t. each pattern
  for (i in 1:ddalpha$numPatterns){
    depth_wrt_i = #calculate depths of the objects, as vector
    depths <- cbind(depths, depth_wrt_i)
  }
  return (depths)
}
ddalpha.train(data, depth = "MyDepth", ...)

```

**To define a classifier:**

**.<NAME>\_validate** validates parameters passed to `ddalpha.train` and passes them to the `ddalpha` object

IN:

`ddalpha` the `ddalpha` object, containing the data and settings (mandatory)  
 <custom parameters> parameters that are passed to the user-defined method  
 ... other parameters (mandatory)

OUT:

`list()` list of output parameters, after the validation is finished, these parameters are stored in the `ddalpha`

**.<NAME>\_learn** trains the classifier. Is different for binary and multiclass classifiers.

IN:

`ddalpha` the `ddalpha` object, containing the data and settings

index1 (only for binary) index of the first class  
 index2 (only for binary) index of the second class  
 depths1 (only for binary) depths of the first class w.r.t. all classes  
 depths2 (only for binary) depths of the second class w.r.t. all classes

depths w.r.t. only given classes are received by `depths1[,c(index1, index2)]`

for the multiclass classifiers the depths are accessible via `ddalpha$patterns[[i]]$depths`

OUT:

`classifier` the trained classifier object

**`<NAME>_classify`** classifies the objects

IN:

`ddalpha` the `ddalpha` object, containing the data and global settings  
`classifier` the previously trained classifier  
`objects` the objects (depths) that are classified

OUT:

`result` a vector with classification results  
 (binary) the objects from class "`classifier$index1`" get positive values  
 (multiclass) the objects get the numbers of patterns in `ddalpha`

Usage: `ddalpha.train(data, separator = "<NAME>", ...)`

#### Custom classifiers ####

```
.MyClassifier_validate <- function(ddalpha, my.parameter = "value", ...){
  print("MyClassifier validating")
  # validate the parameters
  ...
  # always include methodSeparatorBinary.
  # TRUE for the binary classifier, FALSE otherwise
  return(list(methodSeparatorBinary = T,
             my.parameter = my.parameter
             ))
}
```

# a binary classifier

# the package takes care of the voting procedures. Just train it as if there are only two classes

```
.MyClassifier_learn <- function(ddalpha, index1, index2, depths1, depths2){
  print("MyClassifier (binary) learning")
  # The parameters are accessible in the ddalpha object:
  my.parameter = ddalpha$my.parameter
  #depths w.r.t. only given classes are received by
  depths1[,c(index1, index2)]
  depths2[,c(index1, index2)]
  # train the classifier
  classifier <- ...
  #return the needed values in a list, e.g.
  return(list(
```

```

    coefficients = classifier$coefficients,
    ... ))
}
# a multiclass classifier
.MyClassifier_learn <- function(ddalpha){
  print("MyClassifier (multiclass) learning")
  # access the data through the ddalpha object, e.g.
  for (i in 1:ddalpha$numPatterns){
    depth <- ddalpha$patterns[[i]]$depths
    number <- ddalpha$patterns[[i]]$cardinality
    ...
  }
  # train the classifier
  classifier <- ...
  # return the classifier
  return(classifier)
}
# the interface of the classify function is equal for binary and multiclass classifiers
.MyClassifier_classify <- function(ddalpha, classifier, depths){
  print("MyClassifier classifying")
  # The global parameters are accessible in the ddalpha object:
  my.parameter = ddalpha$my.parameter
  # The parameters generated by .MyClassifier_learn are accessible in the classifier object:
  classifier$coefficients
  # here are the depths w.r.t. the first class
  depths[,classifier$index1]
  # here are the depths w.r.t. the second class
  depths[,classifier$index2]
  # fill results in a vector, so that:
  # (binary) the objects from class "classifier$index1" get positive values
  # (multiclass) the objects get the numbers of patterns in ddalpha
  result <- ...
  return(result)
}
ddalpha.train(data, separator = "MyClassifier", ...)

```

**See Also**

[ddalpha.train](#)

**Examples**

```

## Not run:

#### example: Euclidean depth ####

#.Euclidean_validate is basically not needed

.Euclidean_learn <- function(ddalpha){

```

```

print("Euclidean depth learning")

#1. Calculate any statistics based on data that .MyDepth_depths needs
# and store them to the ddalpha object:
for (i in 1:ddalpha$numPatterns){
  ddalpha$patterns[[i]]$center <- colMeans(ddalpha$patterns[[i]]$points)
}

#2. Calculate depths for each pattern
for (i in 1:ddalpha$numPatterns){
  ddalpha$patterns[[i]]$depths = .Euclidian_depths(ddalpha, ddalpha$patterns[[i]]$points)
}

return(ddalpha)
}

.Euclidean_depths <- function(ddalpha, objects){
  print("Euclidean depth calculating")
  depths <- NULL

  #calculate the depths of the objects w.r.t. each pattern
  for (i in 1:ddalpha$numPatterns){
    # The depth parameters are accessible in the ddalpha object:
    center = ddalpha$patterns[[i]]$center

    depth_wrt_i <- 1/(1 + colSums((t(objects) - center)^2))
    depths <- cbind(depths, depth_wrt_i)
  }

  return (depths)
}

#### example: binary decision tree ####

library(rpart)

.tree_validate <- function(ddalpha, ...){
  print("tree validating")
  return(list(methodSeparatorBinary = T))
}

# a binary classifier
# the package takes care of the voting procedures. Just train it as if there are only two classes
.tree_learn <- function(ddalpha, index1, index2, depths1, depths2){
  print("tree learning")

  # prepare the data
  data = as.data.frame(cbind( rbind(depths1, depths2)),
                      c(rep(1, times = nrow(depths1)), rep(-1, times = nrow(depths1))))
  names(data) <- paste0("V", seq_len(ncol(data)))
  names(data)[ncol(data)] <- "O"
  # train the classifier
  classifier <- rpart(O~., data = data)

```

```

    #return the needed values in a list, e.g.
    return(classifier)
}

# the interface of the classify function is equal for binary and multiclass classifiers
.tree_classify <- function(ddalpha, classifier, depths){
  print("tree classifying")

  # fill results in a vector, so that the objects from class "classifier$index1" get positive values
  data = as.data.frame(depths)
  names(data) <- paste0("V",seq_len(ncol(data)))
  result <- predict(classifier, as.data.frame(depths), type = "vector")

  return(result)
}

#### checking ####

library(ddalpha)
data = getdata("hemophilia")

ddalpha = ddalpha.train(data = data, depth = "Euclidean", separator = "tree")
c = ddalpha.classify(ddalpha, data[,1:2])
errors = sum(unlist(c) != data[,3])/nrow(data)
print(paste("Error rate: ",errors))

# building the depth contours using the custom depth
depth.contours.ddalpha(ddalpha, asp = T, levels = seq(0.5, 1, length.out = 10))

## End(Not run)

```

---

dataf

*Converts data from fdata class to the functional class.*


---

### Description

**fd.usc** contains a handy function `fdata` that converts various types of functional data to the `fdata` class. To use these data in `ddalphaf.train` it must first be converted with `dataf`.

The function may be used either to convert a `fdata` object that contains multiple classes, or to convert multiple `fdata` objects, each of which contains one class.

Note that `fdata$fdata2d = TRUE` is not supported.

**Usage**

```
dataf(fdatas, labels)
```

**Arguments**

**fdatas** an fdata object with curves belonging to multiple classes, or a list of fdata objects, each of which contains curves of the same class

**labels** a list of labels of the functional observations. If fdatas is a single fdata object, the list contains labels for each curve. If fdatas is a list of fdata objects, the list labels for each of these fdata objects.

**Format**

The functional data as a data structure (see [dataf.\\*](#)).

**dataf** The functional data as a list of objects. Each object is characterized by two coordinates

**args** The arguments vector

**vals** The values vector

**labels** The classes of the objects

**See Also**

[dataf.\\*](#) for the functional data format.

[ddalphaf.train](#) to train the functional  $DD\alpha$ -classifier

[compclassf.train](#) to train the functional componentwise classifier

[plot.functional](#) for building plots of functional data

**Examples**

```
## Not run:
library(fda.usc)
data(phoneme)

# 1. convert a fdata object that contains multiple classes.
# labels are defined for each curve
converted = dataf(phoneme$learn, phoneme$classlearn)
plot.functional(converted)

# 2. convert multiple fdata objects, each of which contains one class
# the same label is applied to all curves of each fdata object
converted = dataf(list(phoneme$learn, phoneme$test), c("1 red", "2 blue"))
converted$name = "Phoneme learn (red) and test (blue)"
plot.functional(converted)

## End(Not run)
```

---

dataf.*	<i>Functional Data Sets</i>
---------	-----------------------------

---

### Description

The functions generate data sets of functional two-dimensional data of two or more classes.

### Usage

```
# dataf.[name]()  
  
# load the data set by name  
# data(list = "name")  
  
# load the data set by name to a variable  
# getdata("name")
```

### Format

The functional data as a data structure.

dataf The functional data as a list of objects. Each object is characterized by two coordinates

args The arguments vector

vals The values vector

labels The classes of the objects

### Details

More details about the datasets in the topics:

[dataf.geneexp](#)

[dataf.growth](#)

[dataf.medflies](#)

[dataf.population](#)

[dataf.population2010](#)

[dataf.tecator](#)

[dataf.tecator](#)

The following datasets provide simulated data:

[dataf.sim.1.CFF07](#)

[dataf.sim.2.CFF07](#)

### See Also

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the Growth dataset
dataf = dataf.growth()

## view the classes
unique(dataf$labels)

## access the 5th point of the 2nd object
dataf$dataf[[2]]$args[5]
dataf$dataf[[2]]$vals[5]

## Not run: plot.functional(dataf)
```

---

dataf.geneexp

*Gene Expression Profile Data*


---

**Description**

A subset of the *Drosophila* life cycle gene expression data of Arbeitman et al. (2002). The original data set contains 77 gene expression profiles during 58 sequential time points from the embryonic, larval, and pupal periods of the life cycle. The gene expression levels were obtained by a cDNA microarray experiment.

**Usage**

```
dataf.geneexp()
```

**Format**

The functional data as a data structure.

dataf The functional data as a list of objects. Each object is characterized by two coordinates.

args **Time** - a numeric vector of time periods

vals **Gene Expression Level** - a numeric vector

labels Biological classifications identified in Arbeitman et al.(2002) (1 = transient early zygotic genes; 2 = muscle-specific genes; 3 = eye-specific genes. )

**Source**

Chiou, J.-M. and Li, P.-L. Functional clustering and identifying substructures of longitudinal data, *J. R. Statist. Soc. B*, Volume 69 (2007), 679-699

Arbeitman, M.N., Furlong, E.E.M., Imam, F., Johnson, E., Null, B.H., Baker, B.S., Krasnow, M.A., Scott, M.P., Davis, R.W. and White, K.P. (2002) Gene expression during the life cycle of *Drosophila melanogaster*. *Science*, 297, 2270-2274.

**See Also**

[dataf.\\*](#) for other functional data sets

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the dataset
dataf = dataf.geneexp()

## view the classes
unique(dataf$labels)

## access the 5th point of the 2nd object
dataf$dataf[[2]]$args[5]
dataf$dataf[[2]]$vals[5]

## plot the data
## Not run:
labels = unlist(dataf$labels)
plot(dataf,
      xlab="Time", ylab="Gene Expression Level",
      main=paste0("Gene Expression: 1 red (", sum(labels == 1), ")", " ",
                 "2 green (", sum(labels == 2), ")", " ",
                 "3 blue (", sum(labels == 3), ")", ")",
      colors = c("red", "green", "blue"))

## End(Not run)
```

---

dataf.growth

*Berkeley Growth Study Data*

---

**Description**

The data set contains the heights of 39 boys and 54 girls from age 1 to 18 and the ages at which they were collected.

**Usage**

```
dataf.growth()
```

**Format**

The functional data as a data structure.

`dataf` The functional data as a list of objects. Each object is characterized by two coordinates

args **age** - a numeric vector of length 31 giving the ages at which the heights were measured

vals **height** - a numeric vector of heights in centimeters of 39 boys and 54 girls

labels The classes of the objects: boy, girl

**Details**

The ages are not equally spaced.

**Source**

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

Ramsay, James O., and Silverman, Bernard W. (2002), *Applied Functional Data Analysis*, Springer, New York, ch. 6.

Tuddenham, R. D., and Snyder, M. M. (1954) "Physical growth of California boys and girls from birth to age 18", *University of California Publications in Child Development*, 1, 183-364.

**See Also**

[dataf.\\*](#) for other functional data sets

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the Growth dataset
dataf = dataf.growth()

## view the classes
unique(dataf$labels)

## access the 5th point of the 2nd object
dataf$dataf[[2]]$args[5]
dataf$dataf[[2]]$vals[5]

## plot the data
## Not run:
labels = unlist(dataf$labels)
plot(dataf,
      main = paste("Growth: girls red (", sum(labels == "girl"), ")",
                  " boys blue (", sum(labels == "boy"), ")", sep=""),
      xlab="Year", ylab="Height, cm",
      colors = c("blue", "red") # in alphabetical order of class labels
      )

## End(Not run)
```

---

dataf.medflies

*Relationship of Age Patterns of Fecundity to Mortality for Female Medflies.*

---

**Description**

The data set consists of number of eggs laid daily for each of 1000 medflies (Mediterranean fruit flies, *Ceratitis capitata*) until time of death. Data were obtained in Dr. Carey's laboratory. The main questions are to explore the relationship of age patterns of fecundity to mortality, longevity and lifetime reproduction.

A basic finding was that individual mortality is associated with the time-dynamics of the egg-laying trajectory. An approximate parametric model of the egg laying process was developed and used in Muller et al. (2001). Non-parametric approaches which extend principal component analysis for curve data to the situation when covariates are present have been developed and discussed in Chiou, Muller and Wang (2003) and Chiou et al. (2003).

**Usage**

```
dataf.medflies()
```

**Format**

The functional data as a data structure.

`dataf` The functional data as a list of objects. Each object is characterized by two coordinates.

args **day** - a numeric vector of the days numbers

vals **#eggs** - a numeric vector of numbers of eggs laid daily

labels The classes of the objects: long-lived, short-lived

**Source**

Carey, J.R., Liedo, P., Muller, H.G., Wang, J.L., Chiou, J.M. (1998). Relationship of age patterns of fecundity to mortality, longevity, and lifetime reproduction in a large cohort of Mediterranean fruit fly females. *J. of Gerontology –Biological Sciences* 53, 245-251.

Muller, H.G., Carey, J.R., Wu, D., Liedo, P., Vaupel, J.W. (2001). Reproductive potential predicts longevity of female Mediterranean fruit flies. *Proceedings of the Royal Society B* 268, 445-450.

Chiou, J.M., Muller, H.G., Wang, J.L. (2003). Functional quasi-likelihood regression models with smooth random effects. *J. Royal Statist. Soc. B* 65, 405-423.

Chiou, J.M., Muller, H.G., Wang, J.L., Carey, J.R. (2003). A functional multiplicative effects model for longitudinal data, with application to reproductive histories of female medflies. *Statistica Sinica* 13, 1119-1133.

Chiou, J.M., Muller, H.G., Wang, J.L. (2004). Functional response models. *Statistica Sinica* 14, 675-693.

**See Also**

[dataf.\\*](#) for other functional data sets

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the dataset
dataf = dataf.medflies()

## view the classes
unique(dataf$labels)

## access the 5th point of the 2nd object
dataf$dataf[[2]]$args[5]
dataf$dataf[[2]]$vals[5]

## plot the data
## Not run:
labels = unlist(dataf$labels)
plot(dataf,
      xlab="Day", ylab="# eggs",
      main=paste("Medflies (training time):\n short-lived red (", sum(labels == "short-lived"), ")",",",
                " long-lived blue (", sum(labels == "long-lived"), ")",", sep=""),
      colors = c("blue", "red") # in alphabetical order of class labels
      )

## End(Not run)
```

---

dataf.population	<i>World Historical Population-by-Country Dataset</i>
------------------	---

---

**Description**

Historical world population data by countries.

**Usage**

```
dataf.population()
```

**Format**

The functional data as a data structure.

**dataf** The functional data as a list of objects. Each object is characterized by two coordinates

args **year** - a numeric vector of years 1950-2015 (66 years)

vals **population** - a numeric vector of the estimated total population in thousands in 233 countries and regions

**labels** The geographic region of the country: Africa, Asia, Europe, Latin America, North America, Oceania

**identifier** The name of country or region

**Details**

World population data by a country, area or region as of 1 July of the year indicated. Figures are presented in thousands.

**Source**

United Nations, Department of Economic and Social Affairs, Population Division, <https://esa.un.org/unpd/wpp/Download/Standard/Population/>, file Total population –Both sexes

**See Also**

[dataf.population2010](#)

[dataf.\\*](#) for other functional data sets

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the Population dataset
dataf = dataf.population()

## view the classes
unique(dataf$labels)

## access the 5th point of the 2nd object
dataf$dataf[[2]]$args[5]
dataf$dataf[[2]]$vals[5]

## plot the data
## Not run:
labels = unlist(dataf$labels)
plot(dataf,
      main = "World population data",
      xlab="Year", ylab="Population (in thousands)"
      )
## End(Not run)

## compute the integrated and infimal depths of the data curves
## with respect to the same set of curves
depthf.fd1(dataf$dataf, dataf$dataf)
```

---

dataf.population2010 *World Historical Population-by-Country Dataset (2010 Revision)*

---

**Description**

Historical world population data by countries.

**Usage**

```
dataf.population2010()
```

**Format**

The functional data as a data structure.

`dataf` The functional data as a list of objects. Each object is characterized by two coordinates

args **year** - a numeric vector of years 1950-2010 (61 years)

vals **population** - a numeric vector of the estimated total population in thousands in 233 countries and regions

labels The geographic region of the country

identifier The name of country or region

**Details**

World population data by a country, area or region as of 1 July of the year indicated. Figures are presented in thousands.

**Source**

United Nations, Department of Economic and Social Affairs, Population Division, <https://esa.un.org/unpd/wpp/Download/Standard/Population/>, file Total population –Both sexes

**See Also**

[dataf.population](#)

[dataf.\\*](#) for other functional data sets

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the Population dataset
dataf = dataf.population2010()

## view the classes
unique(dataf$labels)

## access the 5th point of the 2nd object
dataf$dataf[[2]]$args[5]
dataf$dataf[[2]]$vals[5]

## plot the data
## Not run:
labels = unlist(dataf$labels)
plot(dataf,
      main = "World population data",
      xlab="Year", ylab="Population (in thousands)"
      )
```

```
## End(Not run)

## compute the integrated and infimal depths of the data curves
## with respect to the same set of curves
depthf.fd1(dataf$dataf, dataf$dataf)
```

---

dataf.sim.1.CFF07      *Model 1 from Cuevas et al. (2007)*

---

### Description

Model 1 from Cuevas et al. (2007)

Processes:

$X(t) = m_0(t) + e(t)$ ,  $m_0(t) = 30*(1-t)*t^{1.2}$

$Y(t) = m_1(t) + e(t)$ ,  $m_1(t) = 30*(1-t)^{1.2}*t$

$e(t)$ : Gaussian with mean = 0,  $cov(X(s), X(t)) = 0.2*\exp(-abs(s - t)/0.3)$

the processes are discretized at numDiscrets equally distant points on [0, 1]. The functions are smooth and differ in mean only, which makes the classification task rather simple.

### Usage

```
dataf.sim.1.CFF07(numTrain = 100, numTest = 50, numDiscrets = 51, plot = FALSE)
```

### Arguments

numTrain	number of objects in the training sample
numTest	number of objects in the test sample
numDiscrets	number of points for each object
plot	if TRUE the training sample is plotted

### Format

A data structure containing \$learn and \$test functional data. The functional data are given as data structures.

dataf The functional data as a list of objects. Each object is characterized by two coordinates.

args a numeric vector

vals a numeric vector

labels The classes of the objects: 0 for X(t), 1 for Y(t)

### Source

Cuevas, A., Febrero, M. and Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions. Computational Statistics 22 481-496.

**See Also**

[dataf.\\*](#) for other functional data sets

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the dataset
dataf = dataf.sim.1.CFF07(numTrain = 100, numTest = 50, numDiscrets = 51)
learn = dataf$learn
test = dataf$test

## view the classes
unique(learn$labels)

## access the 5th point of the 2nd object
learn$dataf[[2]]$args[5]
learn$dataf[[2]]$vals[5]

## Not run:
## plot the data
plot(learn)
plot(test)

## or
dataf = dataf.sim.1.CFF07(numTrain = 100, numTest = 50, numDiscrets = 51, plot = TRUE)

## End(Not run)
```

---

dataf.sim.2.CFF07      *Model 2 from Cuevas et al. (2007)*

---

**Description**

Model 2 from Cuevas et al. (2007)

Processes:

$X(t) = m_0(t) + e(t)$ ,  $m_0(t) = 30*(1-t)*t^2 + 0.5*abs(sin(20*pi*t))$

$Y(t)$  = an 8-knot spline approximation of  $X$

$e(t)$ : Gaussian with mean = 0,  $cov(X(s), X(t)) = 0.2*exp(-abs(s - t)/0.3)$

the processes are discretized at numDiscrets equally distant points on [0, 1].

**Usage**

```
dataf.sim.2.CFF07(numTrain = 100, numTest = 50, numDiscrets = 51, plot = FALSE)
```

**Arguments**

numTrain	number of objects in the training sample
numTest	number of objects in the test sample
numDiscrets	number of points for each object
plot	if TRUE the training sample is plotted

**Format**

A data structure containing \$learn and \$test functional data. The functional data are given as data structures.

dataf The functional data as a list of objects. Each object is characterized by two coordinates.

args a numeric vector

vals a numeric vector

labels The classes of the objects: 0 for X(t), 1 for Y(t)

**Source**

Cuevas, A., Febrero, M. and Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics* 22 481-496.

**See Also**

[dataf.\\*](#) for other functional data sets

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the dataset
dataf = dataf.sim.2.CFF07(numTrain = 100, numTest = 50, numDiscrets = 51)
learn = dataf$learn
test = dataf$test

## view the classes
unique(learn$labels)

## access the 5th point of the 2nd object
learn$dataf[[2]]$args[5]
learn$dataf[[2]]$vals[5]

## Not run:
## plot the data
plot(learn)
plot(test)

## or
dataf = dataf.sim.2.CFF07(numTrain = 100, numTest = 50, numDiscrets = 51, plot = TRUE)

## End(Not run)
```

---

`dataf.tecator`*Functional Data Set Spectrometric Data (Tecator)*

---

### Description

This dataset is a part of the original one which can be found at <https://www.stat.cmu.edu/>. For each piece of finely chopped meat we observe one spectrometric curve which corresponds to the absorbance measured at 100 wavelengths. The pieces are split according to Ferraty and Vieu (2006) into two classes: with small (<20) and large fat content obtained by an analytical chemical processing.

### Usage

```
dataf.tecator()
```

### Format

The functional data as a data structure.

`dataf` The functional data as a list of objects. Each object is characterized by two coordinates.

args **wavelength** - a numeric vector of discretization points from 850 to 1050mm

vals **absorbance** - a numeric vector of absorbance values

labels The classes of the objects: "small" (<20) and "large" fat content

### Author(s)

Febrero-Bande, M and Oviedo de la Fuente, Manuel

### Source

<https://www.stat.cmu.edu/>

### References

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis: theory and practice*. Springer.

### See Also

[dataf.\\*](#) for other functional data sets

[plot.functional](#) for building plots of functional data

**Examples**

```
## load the dataset
dataf = dataf.tecator()

## view the classes
unique(dataf$labels)

## access the 5th point of the 2nd object
dataf$dataf[[2]]$args[5]
dataf$dataf[[2]]$vals[5]

## plot the data
## Not run:
labels = unlist(dataf$labels)
plot(dataf,
      xlab="Wavelengths", ylab="Absorbances",
      main=paste("Tecator: < 20 red (", sum(labels == "small"), ")",
                " >= 20 blue (", sum(labels == "large"), ")", sep=""),
      colors = c("blue", "red"))

## End(Not run)
```

---

dataf2rawfd

*Transform a dataf Object to Raw Functional Data*


---

**Description**

From a (possibly multivariate) functional data object `dataf` constructs an array of the functional values evaluated at an equi-distant grid of points.

**Usage**

```
dataf2rawfd(dataf, range = NULL, d = 101)
```

**Arguments**

<code>dataf</code>	Functions to be transformed, represented by a (possibly multivariate) <code>dataf</code> object of their arguments and functional values. <code>m</code> stands for the number of functions. The grid of observation points for the functions in <code>dataf</code> may not be the same.
<code>range</code>	The common range of the domain where the functions <code>dataf</code> are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in <code>dataf</code> . If the range is not provided, the smallest interval in which all the arguments from the data functions are contained is chosen as the domain.

**d** Grid size to which all the functional data are transformed. All functional observations are transformed into vectors of their functional values of length  $d$  corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation, see Nagy et al. (2016).

### Value

If the functional data are univariate (scalar-valued), a matrix of size  $m \times d$  is given, with each row corresponding to one function. If the functional data are  $k$ -variate with  $k > 1$ , an array of size  $m \times d \times k$  of the functional values is given.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### See Also

[rawfd2dataf](#)

[depthf.fd1](#)

[depthf.fd2](#)

### Examples

```
## transform a matrix into a functional data set and back
n = 5
d = 21
X = matrix(rnorm(n*d), ncol=d)
R = rawfd2dataf(X, range=c(0,1))
R2 = dataf2rawfd(R, range=c(0,1), d=d)
all.equal(X, R2)

## transform a functional dataset into a raw matrix of functional values
dataf = dataf.population()$dataf
dataf2rawfd(dataf, range=c(1950,2015), d=66)

## transform an array into a multivariate functional data set and back
k = 3
X = array(rnorm(n*d*k), dim=c(n,d,k))
R = rawfd2dataf(X, range=c(-1,1))
dataf2rawfd(R, range=c(-1,1), d=50)
```

---

ddalpha.classify      *Classify using DD-Classifier*

---

### Description

Classifies data using the DD-classifier and a specified outsider treatment.

### Usage

```
ddalpha.classify(ddalpha, objects, subset, outsider.method = NULL, use.convex = NULL)
```

```
## S3 method for class 'ddalpha'
predict(object, objects, subset, outsider.method = NULL, use.convex = NULL, ...)
```

### Arguments

ddalpha, object	DD $\alpha$ -classifier (obtained by <a href="#">ddalpha.train</a> ).
objects	Matrix containing objects to be classified; each row is one $d$ -dimensional object.
subset	an optional vector specifying a subset of observations to be classified.
outsider.method	Character string, name of a treatment to be used for outsiders; one of those trained by <a href="#">ddalpha.train</a> . If the treatment was specified using the argument <code>outsider.methods</code> then use the name of the method.
use.convex	Logical variable indicating whether outsiders should be determined as the points not contained in any of the convex hulls of the classes from the training sample (TRUE) or those having zero depth w.r.t. each class from the training sample (FALSE). For <code>depth = "zonoid"</code> both values give the same result. If NULL the value specified in DD $\alpha$ -classifier (in <a href="#">ddalpha.train</a> ) is used.
...	additional parameters are ignored

### Details

Only one outsider treatment can be specified.

See Lange, Mosler and Mozharovskyi (2014) for details and additional information.

### Value

List containing class labels, or character string "Ignored" for the outsiders if "Ignore" was specified as the outsider treating method.

## References

- Dyckerhoff, R., Koshevoy, G., and Mosler, K. (1996). Zonoid data depth: theory and computation. In: Prat A. (ed), *COMPSTAT 1996. Proceedings in computational statistics*, Physica-Verlag (Heidelberg), 235–240.
- Lange, T., Mosler, K., and Mozharovskiy, P. (2014). Fast nonparametric classification based on data depth. *Statistical Papers* **55** 49–69.
- Li, J., Cuesta-Albertos, J.A., and Liu, R.Y. (2012). DD-classifier: Nonparametric classification procedure based on DD-plot. *Journal of the American Statistical Association* **107** 737–753.
- Mozharovskiy, P. (2015). *Contributions to Depth-based Classification and Computation of the Tukey Depth*. Verlag Dr. Kovac (Hamburg).
- Mozharovskiy, P., Mosler, K., and Lange, T. (2015). Classifying real-world data with the DD $\alpha$ -procedure. *Advances in Data Analysis and Classification* **9** 287–314.
- Vasil'ev, V.I. (2003). The reduction principle in problems of revealing regularities I. *Cybernetics and Systems Analysis* **39** 686–694.

## See Also

[ddalpha.train](#) to train the DD-classifier.

## Examples

```
# Generate a bivariate normal location-shift classification task
# containing 200 training objects and 200 to test with
class1 <- mvrnorm(200, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
trainIndices <- c(1:100)
testIndices <- c(101:200)
propertyVars <- c(1:2)
classVar <- 3
trainData <- rbind(cbind(class1[trainIndices,], rep(1, 100)),
                  cbind(class2[trainIndices,], rep(2, 100)))
testData <- rbind(cbind(class1[testIndices,], rep(1, 100)),
                  cbind(class2[testIndices,], rep(2, 100)))
data <- list(train = trainData, test = testData)

# Train the DDalpha-Classifier (zonoid depth, maximum Mahalanobis depth
# classifier with defaults as outsider treatment)
ddalpha <- ddalpha.train(data$train,
                        depth = "zonoid",
                        outsider.methods = "depth.Mahalanobis")

# Get the classification error rate
classes <- ddalpha.classify(data$test[,propertyVars], ddalpha,
                           outsider.method = "depth.Mahalanobis")
cat("Classification error rate: ",
    sum(unlist(classes) != data$test[,classVar])/200, ".\n", sep="")
```

---

 ddalpha.getErrorRateCV

*Test DD-Classifier*


---

### Description

Performs a cross-validation procedure over the given data. On each step every numchunks observation is removed from the data, the DD-classifier is trained on these data and tested on the removed observations.

### Usage

```
ddalpha.getErrorRateCV (data, numchunks = 10, ...)
```

### Arguments

data	Matrix containing training sample where each of $n$ rows is one object of the training sample where first $d$ entries are inputs and the last entry is output (class label).
numchunks	number of subsets of testing data. Equals to the number of times the classifier is trained.
...	additional parameters passed to <a href="#">ddalpha.train</a>

### Value

errors	the part of incorrectly classified data
time	the mean training time
time_sd	the standard deviation of training time

### See Also

[ddalpha.train](#) to train the  $DD_\alpha$ -classifier, [ddalpha.classify](#) for classification using  $DD_\alpha$ -classifier, [ddalpha.test](#) to test the DD-classifier on particular learning and testing data, [ddalpha.getErrorRatePart](#) to perform a benchmark study of the DD-classifier on particular data.

### Examples

```
# Generate a bivariate normal location-shift classification task
# containing 200 training objects and 200 to test with
class1 <- mvrnorm(150, c(0,0),
  matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(150, c(2,2),
  matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
propertyVars <- c(1:2)
classVar <- 3
data <- rbind(cbind(class1, rep(1, 150)), cbind(class2, rep(2, 150)))
```

```

# Train 1st DDalpha-classifier (default settings)
# and get the classification error rate
stat <- ddalpha.getErrorRateCV(data, numchunks = 5)
cat("1. Classification error rate (defaults): ",
    stat$error, ".\n", sep = "")

# Train 2nd DDalpha-classifier (zonoid depth, maximum Mahalanobis
# depth classifier with defaults as outsider treatment)
# and get the classification error rate
stat2 <- ddalpha.getErrorRateCV(data, depth = "zonoid",
                                outsider.methods = "depth.Mahalanobis")
cat("2. Classification error rate (depth.Mahalanobis): ",
    stat2$error, ".\n", sep = "")

```

---

```
ddalpha.getErrorRatePart
```

*Test DD-Classifier*

---

## Description

Performs a benchmark procedure by partitioning the given data. On each of times steps size observations are removed from the data, the DD-classifier is trained on these data and tested on the removed observations.

## Usage

```
ddalpha.getErrorRatePart(data, size = 0.3, times = 10, ...)
```

## Arguments

data	Matrix containing training sample where each of $n$ rows is one object of the training sample where first $d$ entries are inputs and the last entry is output (class label).
size	the excluded sequences size. Either an integer between 1 and $n$ , or a fraction of data between 0 and 1.
times	the number of times the classifier is trained.
...	additional parameters passed to <code>ddalpha.train</code>

## Value

errors	the part of incorrectly classified data (mean)
errors_sd	the standard deviation of errors
errors_vec	vector of errors
time	the mean training time
time_sd	the standard deviation of training time

**See Also**

[ddalpha.train](#) to train the  $DD_\alpha$ -classifier, [ddalpha.classify](#) for classification using  $DD_\alpha$ -classifier, [ddalpha.test](#) to test the DD-classifier on particular learning and testing data, [ddalpha.getErrorRateCV](#) to get error rate of the DD-classifier on particular data.

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 200 objects
class1 <- mvrnorm(100, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(100, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
propertyVars <- c(1:2)
classVar <- 3
data <- rbind(cbind(class1, rep(1, 100)), cbind(class2, rep(2, 100)))

# Train 1st DDalpha-classifier (default settings)
# and get the classification error rate
stat <- ddalpha.getErrorRatePart(data, size = 10, times = 10)
cat("1. Classification error rate (defaults): ",
    stat$error, ".\n", sep = "")

# Train 2nd DDalpha-classifier (zonoid depth, maximum Mahalanobis
# depth classifier with defaults as outsider treatment)
# and get the classification error rate
stat2 <- ddalpha.getErrorRatePart(data, depth = "zonoid",
                                 outsider.methods = "depth.Mahalanobis", size = 0.2, times = 10)
cat("2. Classification error rate (depth.Mahalanobis): ",
    stat2$error, ".\n", sep = "")
```

---

ddalpha.test

*Test DD-Classifier*


---

**Description**

Trains DD-classifier on the learning sequence of the data and tests it on the testing sequence.

**Usage**

```
ddalpha.test(learn, test, ...)
```

**Arguments**

learn	the learning sequence of the data. Matrix containing training sample where each of $n$ rows is one object of the training sample where first $d$ entries are inputs and the last entry is output (class label).
test	the testing sequence. Has the same format as learn
...	additional parameters passed to <a href="#">ddalpha.train</a>

**Value**

error	the part of incorrectly classified data
correct	the number of correctly classified objects
incorrect	the number of incorrectly classified objects
total	the number of classified objects
ignored	the number of ignored objects (outside the convex hull of the learning data)
n	the number of objects in the testing sequence
time	training time

**See Also**

[ddalpha.train](#) to train the DD-classifier, [ddalpha.classify](#) for classification using DD-classifier, [ddalpha.getErrorRateCV](#) and [ddalpha.getErrorRatePart](#) to get error rate of the DD-classifier on particular data.

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 200 training objects and 200 to test with
class1 <- mvrnorm(200, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
trainIndices <- c(1:100)
testIndices <- c(101:200)
propertyVars <- c(1:2)
classVar <- 3
trainData <- rbind(cbind(class1[trainIndices,], rep(1, 100)),
                  cbind(class2[trainIndices,], rep(2, 100)))
testData <- rbind(cbind(class1[testIndices,], rep(1, 100)),
                  cbind(class2[testIndices,], rep(2, 100)))
data <- list(train = trainData, test = testData)

# Train 1st DDalpha-classifier (default settings)
# and get the classification error rate
stat <- ddalpha.test(data$train, data$test)
cat("1. Classification error rate (defaults): ",
    stat$error, ".\n", sep = "")
```

```
# Train 2nd DDalpha-classifier (zonoid depth, maximum Mahalanobis
# depth classifier with defaults as outsider treatment)
# and get the classification error rate
stat2 <- ddalpha.test(data$train, data$test, depth = "zonoid",
                     outsider.methods = "depth.Mahalanobis")
cat("2. Classification error rate (depth.Mahalanobis): ",
    stat2$error, ".\n", sep = "")
```

---

ddalpha.train

*Train DD-Classifier*


---

### Description

Trains the DD-classifier using a training sample according to given parameters. The DD-classifier is a non-parametric procedure that first transforms the training sample into the depth space calculating the depth of each point w.r.t each class (dimension of this space equals the number of classes in the training sample), and then constructs a separating rule in this depth space. If in the classification phase an object does not belong to the convex hull of at least one class (we mention such an object as an 'outsider'), it is mapped into the origin of the depth space and hence cannot be classified in the depth space. For these objects, after 'outsiderness' has been assured, an outsider treatment, i.e. a classification procedure functioning outside convex hulls of the classes is applied; it has to be trained too.

The current realization of the DD-classifier allows for several alternative outsider treatments; they involve different traditional classification methods, see 'Details' and 'Arguments' for parameters needed.

The function allows for classification with  $q \geq 2$  classes, see `aggregation.method` in 'Arguments'.

### Usage

```
ddalpha.train(formula, data, subset,
              depth = "halfspace",
              separator = "alpha",
              outsider.methods = "LDA",
              outsider.settings = NULL,
              aggregation.method = "majority",
              pretransform = NULL,
              use.convex = FALSE,
              seed = 0,
              ...)
```

### Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model. If not found in data, the variables of the model are taken from environment.
---------	--

data	<p>Matrix or data.frame containing training sample where each of <math>n</math> rows is one object of the training sample where first <math>d</math> entries are inputs and the last entry is output (class label).</p> <p>A pre-calculated DD-plot may be used as data with <code>depth="ddplot"</code>.</p>
subset	<p>an optional vector specifying a subset of observations to be used in training the classifier.</p>
depth	<p>Character string determining which depth notion to use; the default value is "halfspace". The list of the supported depths is given in section Depths. To use a custom depth, see topic <a href="#">Custom Methods</a>. To use an outsider treatment only set <code>depth = NULL</code>.</p>
separator	<p>The method used for separation on the DD-plot; can be "alpha" (the default), "polynomial", "knnlm" or "maxD". See section Separators for the description of the separators and additional parameters. To use a custom separator, see topic <a href="#">Custom Methods</a>.</p>
outsider.methods	<p>Vector of character strings each being a name of a basic outsider method for eventual classification; possible names are: "LDA" (the default), "QDA", "kNN", "kNNAff", "depth.Mahalanobis", "RandProp", "RandEqual" and "Ignore". Each method can be specified only once, replications are ignored. By specifying treatments in such a way only a basic treatment method can be chosen (by the name), and the default settings for each of the methods are applied, see 'Details'.</p>
outsider.settings	<p>List containing outsider treatments each described by a list of parameters including a name, see 'Details' and 'Examples'. Each method can be used multiply with (not necessarily) different parameters, just the name should be unique, entries with the repeating names are ignored.</p>
aggregation.method	<p>Character string determining which method to apply to aggregate binary classification results during multiclass classification; can be "majority" (the default) or "sequent". If "majority", <math>q(q-1)/2</math> (with <math>q</math> being the number of classes in the training sample) binary classifiers are trained, the classification results are aggregated using the majority voting, where classes with larger proportions in the training sample (eventually with the earlier entries in the data) are preferred when tied. If "sequent", <math>q</math> binary 'one against all'-classifiers are trained and ties during the classification are resolved as before.</p>
pretransform	<p>indicates if the data has to be scaled before the learning procedure. If the used depth method is affine-invariant and pretransform doesn't influence the result, the data won't be transformed (the parameter is ignored).</p> <p><b>NULL</b> applies no transformation to the data</p> <p><b>"1Mom", "1MCD"</b> the data is transformed with the common covariance matrix of the whole data</p> <p><b>"NMom", "NMCD"</b> the data is transformed w.r.t. each class using its covariance matrix. The depths w.r.t. each class are calculated using the transformed data.</p> <p>for the values "1MCD", "NMCD" <code>covMcd</code> is used to calculate the covariance matrix, and the parameter <code>mah.parMcd</code> is used.</p>

use.convex	Logical variable indicating whether outsiders should be determined exactly, i.e. as the points not contained in any of the convex hulls of the classes from the training sample (TRUE), or those having zero depth w.r.t. each class from the training sample (FALSE). For depth = "zonoid" both values give the same result.
seed	the random seed. The default value seed=0 makes no changes.
...	The parameters for the depth calculating and separation methods.

## Details

### Depths:

For depth="ddplot" the pre-calculated DD-plot shall be passed as data.

To use a custom depth, see topic [Custom Methods](#).

To use an outsider treatment only set depth = NULL.

The following depths are supported:

[depth.halfspace](#) for calculation of the Tukey depth.

[depth.Mahalanobis](#) for calculation of Mahalanobis depth.

[depth.projection](#) for calculation of projection depth.

[depth.simplicial](#) for calculation of simplicial depth.

[depth.simplicialVolume](#) for calculation of simplicial volume depth.

[depth.spatial](#) for calculation of spatial depth.

[depth.zonoid](#) for calculation of zonoid depth.

The additional parameters are described in the corresponding topics.

### Separators:

The separators classify data on the 2-dimensional space of a DD-plot built using the depths.

To use a custom separator, see topic [Custom Methods](#).

#### *alpha:*

Trains the  $DD_\alpha$ -classifier (Lange, Mosler and Mozharovskyi, 2014; Mozharovskyi, Mosler and Lange, 2015). The  $DD_\alpha$ -classifier constructs a linear separating rule in the polynomial extension of the depth space with the  $\alpha$ -procedure (Vasil'ev, 2003); maximum degree of the polynomial products is determined via cross-validation (in the depth space).

The additional parameters:

**max.degree** Maximum of the range of degrees of the polynomial depth space extension over which the  $\alpha$ -procedure is to be cross-validated; can be 1, 2 or 3 (default).

**num.chunks** Number of chunks to split data into when cross-validating the  $\alpha$ -procedure; should be  $> 0$ , and smaller than the total number of points in the two smallest classes when `aggregation.method = "majority"` and smaller than the total number of points in the training sample when `aggregation.method = "sequent"`. The default value is 10.

#### *polynomial:*

Trains the polynomial DD-classifier (Li, Cuesta-Albertos and Liu, 2012). The DD-classifier constructs a polynomial separating rule in the depth space; the degree of the polynomial is determined via cross-validation (in the depth space).

The additional parameters:

**max.degree** Maximum of the range of degrees of the polynomial over which the separator is to be cross-validated; can be in [1:10], the default value is 3.

**num.chunks** Number of chunks to split data into when cross-validating the separator; should be  $> 0$ , and smaller than the total number of points in the two smallest classes when `aggregation.method = "majority"` and smaller than the total number of points in the training sample when `aggregation.method = "sequent"`. The default value is 10.

*knnlm*:

Trains the  $k$ -nearest neighbours classifier in the depth space.

The additional parameters:

**knnrange** The maximal number of neighbours for  $k$ NN separation. The value is bounded by 2 and  $n/2$ .

NULL for the default value  $10 * (n^{1/q}) + 1$ , where  $n$  is the number of objects,  $q$  is the number of classes.

"MAX" for the maximum value  $n/2$

*maxD*: The maximum depth separator classifies an object to the class that provides it the largest depth value.

### Outsider treatment:

An outsider treatment is a supplementary classifier for data that lie outside the convex hulls of all  $q$  training classes. Available methods are: Linear Discriminant Analysis (referred to as "LDA", see [lda](#);  $k$ -Nearest-Neighbor Classifier ("kNN"), see [knn](#), [knn.cv](#); Affine-Invariant  $k$ NN ("kNNAff"), an affine-invariant version of the  $k$ NN, suited only for binary classification (some aggregation is used with multiple classes) and not accounting for ties (at all), but very fast by that; Maximum Mahalanobis Depth Classifier ("depth.Mahalanobis"), the outsider is referred to a class w.r.t. which it has the highest depth value scaled by (approximated) priors; Proportional Randomization ("RandProp"), the outsider is referred to a class randomly with probability equal to it (approximated) prior; Equal Randomization ("RandEqual"), the outsider is referred to a class randomly, chances for each class are equal; Ignoring ("Ignore"), the outsider is not classified, the string "Ignored" is returned instead.

An outsider treatment is specified by a list containing a name and parameters:

`name` is a character string, name of the outsider treatment to be freely specified; should be unique; is obligatory.

`method` is a character string, name of the method to use, can be "LDA", "kNN", "kNNAff", "depth.Mahalanobis", "RandProp", "RandEqual" and "Ignore"; is obligatory.

`priors` is a numerical vector specifying prior probabilities of classes; class portions in the training sample are used by the default. `priors` is used in methods "LDA", "depth.Mahalanobis" and "RandProp".

`knn.k` is the number of the nearest neighbors taken into account; can be between 1 and the number of points in the training sample. Set to  $-1$  (the default) to be determined by the leave-one-out cross-validation. `knn.k` is used in method "kNN".

`knn.range` is the upper bound on the range over which the leave-one-out cross-validation is performed (the lower bound is 1); can be between 2 and the number of points in the training sample  $-1$ . Set to  $-1$  (the default) to be calculated automatically accounting for number of points and dimension. `knn.range` is used in method "kNN".

`knnAff.methodAggregation` is a character string specifying the aggregation technique for method "kNNAff"; works in the same way as the function argument `aggregation.method`. `knnAff.methodAggregation` is used in method "kNNAff".

`knnAff.k` is the number of the nearest neighbors taken into account; should be at least 1 and up to the number of points in the training sample when `knnAff.methodAggregation = "sequent"`, and up to the total number of points in the training sample when `knnAff.methodAggregation = "majority"`. Set to `-1` (the default) to be determined by the leave-one-out cross-validation. `knnAff.k` is used in method "kNNAff".

`knnAff.range` is the upper bound on the range over which the leave-one-out cross-validation is performed (the lower bound is 1); should be `> 1` and smaller than the total number of points in the two smallest classes when `knnAff.methodAggregation = "majority"`, and `> 1` and smaller than the total number of points in the training sample when `knnAff.methodAggregation = "sequent"`. Set to `-1` to be calculated automatically accounting for number of points and dimension. `knnAff.range` is used in method "kNNAff".

`mah.estimate` is a character string specifying which estimates to use when calculating the Mahalanobis depth; can be "moment" or "MCD", determining whether traditional moment or Minimum Covariance Determinant (MCD) (see `covMcd`) estimates for mean and covariance are used. `mah.estimate` is used in method "depth.Mahalanobis".

`mcd.alpha` is the value of the argument `alpha` for the function `covMcd`; is used in method "depth.Mahalanobis" when `mah.estimate = "MCD"`.

## Value

Trained  $DD\alpha$ -classifier containing following - rather informative - fields:

<code>num.points</code>	Total number of points in the training sample.
<code>dimension</code>	Dimension of the original space.
<code>depth</code>	Character string determining which depth notion to use.
<code>methodAggregation</code>	Character string determining which method to apply to aggregate binary classification results.
<code>num.chunks</code>	Number of chunks data has been split into when cross-validating the $\alpha$ -procedure.
<code>num.directions</code>	Number of directions used for approximating the Tukey depth (when it is used).
<code>use.convex</code>	Logical variable indicating whether outsiders should be determined exactly when classifying.
<code>max.degree</code>	Maximum of the range of degrees of the polynomial depth space extension over which the $\alpha$ -procedure has been cross-validated.
<code>patterns</code>	Classes of the training sample.
<code>num.classifiers</code>	Number of binary classifiers trained.
<code>outsider.methods</code>	Treatments to be used to classify outsiders.

## References

Dyckerhoff, R., Koshevoy, G., and Mosler, K. (1996). Zonoid data depth: theory and computation. In: Prat A. (ed), *COMPSTAT 1996. Proceedings in computational statistics*, Physica-Verlag (Heidelberg), 235–240.

Lange, T., Mosler, K., and Mozharovskyi, P. (2014). Fast nonparametric classification based on data depth. *Statistical Papers* **55** 49–69.

Li, J., Cuesta-Albertos, J.A., and Liu, R.Y. (2012). DD-classifier: Nonparametric classification procedure based on DD-plot. *Journal of the American Statistical Association* **107** 737–753.

Mozharovskyi, P. (2015). *Contributions to Depth-based Classification and Computation of the Tukey Depth*. Verlag Dr. Kovac (Hamburg).

Mozharovskyi, P., Mosler, K., and Lange, T. (2015). Classifying real-world data with the DD $\alpha$ -procedure. *Advances in Data Analysis and Classification* **9** 287–314.

Vasil'ev, V.I. (2003). The reduction principle in problems of revealing regularities I. *Cybernetics and Systems Analysis* **39** 686–694.

### See Also

[ddalpha.classify](#) for classification using DD-classifier, [depth.](#) for calculation of depths, [depth.space.](#) for calculation of depth spaces, [is.in.convex](#) to check whether a point is not an outsider.

### Examples

```
# Generate a bivariate normal location-shift classification task
# containing 200 training objects and 200 to test with
class1 <- mvrnorm(200, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
trainIndices <- c(1:100)
testIndices <- c(101:200)
propertyVars <- c(1:2)
classVar <- 3
trainData <- rbind(cbind(class1[trainIndices,], rep(1, 100)),
                  cbind(class2[trainIndices,], rep(2, 100)))
testData <- rbind(cbind(class1[testIndices,], rep(1, 100)),
                  cbind(class2[testIndices,], rep(2, 100)))
data <- list(train = trainData, test = testData)

# Train 1st DDalpha-classifier (default settings)
# and get the classification error rate
ddalpha1 <- ddalpha.train(data$train)
classes1 <- ddalpha.classify(ddalpha1, data$test[,propertyVars])
cat("1. Classification error rate (defaults): ",
    sum(unlist(classes1) != data$test[,classVar])/200, ".\n", sep = "")

# Train 2nd DDalpha-classifier (zonoid depth, maximum Mahalanobis
# depth classifier with defaults as outsider treatment)
# and get the classification error rate
ddalpha2 <- ddalpha.train(data$train, depth = "zonoid",
                          outsider.methods = "depth.Mahalanobis")
classes2 <- ddalpha.classify(ddalpha2, data$test[,propertyVars],
                             outsider.method = "depth.Mahalanobis")
cat("2. Classification error rate (depth.Mahalanobis): ",
    sum(unlist(classes2) != data$test[,classVar])/200, ".\n", sep = "")
```

```

# Train 3rd DDalpha-classifier (100 random directions for the Tukey depth,
# adjusted maximum Mahalanobis depth classifier
# and equal randomization as outsider treatments)
# and get the classification error rates
treatments <- list(list(name = "mahd1", method = "depth.Mahalanobis",
                      mah.estimate = "MCD", mcd.alpha = 0.75, priors = c(1, 1)/2),
                  list(name = "rand1", method = "RandEqual"))
ddalpha3 <- ddalpha.train(data$train, outsider.settings = treatments,
                        num.direction = 100)
classes31 <- ddalpha.classify(ddalpha3, data$test[,propertyVars],
                             outsider.method = "mahd1")
classes32 <- ddalpha.classify(ddalpha3, data$test[,propertyVars],
                             outsider.method = "rand1")
cat("3. Classification error rate (by treatments):\n")
cat("  Error (mahd1): ",
    sum(unlist(classes31) != data$test[,classVar])/200, ".\n", sep = "")
cat("  Error (rand1): ",
    sum(unlist(classes32) != data$test[,classVar])/200, ".\n", sep = "")

# Train using some weird formula
ddalpha = ddalpha.train(
  I(mpg >= 19.2) ~ log(displacement) + I(displacement^2) + displacement + I(displacement * drat),
  data = mtcars, subset = (carb!=1),
  depth = "Mahalanobis", separator = "alpha")
print(ddalpha) # make sure that the resulting table is what you wanted
CC = ddalpha.classify(ddalpha, mtcars)
sum((mtcars$mpg>=19.2)!= unlist(CC))/nrow(mtcars) # error rate

#Use the pre-calculated DD-plot
data = cbind(rbind(mvrnorm(n = 50, mu = c(0,0), Sigma = diag(2)),
                  mvrnorm(n = 50, mu = c(5,10), Sigma = diag(2)),
                  mvrnorm(n = 50, mu = c(10,0), Sigma = diag(2))),
            rep(c(1,2,3), each = 50))
plot(data[,1:2], col = (data[,3]+1))

ddplot = depth.space.Mahalanobis(data = data[,1:2], cardinalities = c(50,50,50))
ddplot = cbind(ddplot, data[,3])
ddalphaD = ddalpha.train(data = ddplot, depth = "ddplot", separator = "alpha")
c = ddalpha.classify(ddalphaD, ddplot[,1:3])
errors = sum(unlist(c) != data[,3])/nrow(data)
print(paste("Error rate: ",errors))

ddalpha = ddalpha.train(data = data, depth = "Mahalanobis", separator = "alpha")
c = ddalpha.classify(ddalpha, data[,1:2])
errors = sum(unlist(c) != data[,3])/nrow(data)
print(paste("Error rate: ",errors))

```

**Description**

Classifies data using the functional DD-classifier.

**Usage**

```
ddalphaf.classify(ddalphaf, objectsf, subset, ...)
```

```
## S3 method for class 'ddalphaf'
predict(object, objectsf, subset, ...)
```

**Arguments**

ddalphaf, object	Functional DD-classifier (obtained by <a href="#">ddalphaf.train</a> ).
objectsf	list containing lists (functions) of two vectors of equal length, named "args" and "vals": arguments sorted in ascending order and corresponding them values respectively
subset	an optional vector specifying a subset of observations to be classified.
...	additional parameters, passed to the classifier, selected with parameter <code>classifier.type</code> in <a href="#">ddalphaf.train</a> .

**Value**

List containing class labels.

**References**

Mosler, K. and Mozharovskyi, P. (2017). Fast DD-classification of functional data. *Statistical Papers* **58** 1055–1089.

Mozharovskyi, P. (2015). *Contributions to Depth-based Classification and Computation of the Tukey Depth*. Verlag Dr. Kovac (Hamburg).

**See Also**

[ddalphaf.train](#) to train the functional  $DD\alpha$ -classifier.

**Examples**

```
## Not run:
## load the Growth dataset
dataf = dataf.growth()

learn = c(head(dataf$dataf, 49), tail(dataf$dataf, 34))
labels= c(head(dataf$labels, 49), tail(dataf$labels, 34))
test = tail(head(dataf$dataf, 59), 10) # elements 50:59. 5 girls, 5 boys

c = ddalphaf.train (learn, labels, classifier.type = "ddalpha")
```

```

classified = ddalphaf.classify(c, test)

print(unlist(classified))

## End(Not run)

```

---

```

ddalphaf.getErrorRateCV
      Test Functional DD-Classifier

```

---

### Description

Performs a cross-validation procedure over the given data. On each step every numchunks observation is removed from the data, the functional DD-classifier is trained on these data and tested on the removed observations.

### Usage

```
ddalphaf.getErrorRateCV (dataf, labels, numchunks = 10, disc.type = c("LS", "comp"), ...)
```

### Arguments

dataf	list containing lists (functions) of two vectors of equal length, named "args" and "vals": arguments sorted in ascending order and corresponding them values respectively
labels	list of output labels of the functional observations
numchunks	number of subsets of testing data. Equals to the number of times the classifier is trained.
disc.type	type of the used discretization scheme. "LS" for <a href="#">ddalphaf.train</a> , "comp" for <a href="#">compclassf.train</a>
...	additional parameters passed to <a href="#">ddalphaf.train</a>

### Value

errors	the part of incorrectly classified data
time	the mean training time
time_sd	the standard deviation of training time

### See Also

[ddalphaf.train](#) to train the functional  $DD\alpha$ -classifier, [ddalphaf.classify](#) for classification using functional  $DD\alpha$ -classifier, [ddalphaf.test](#) to test the functional DD-classifier on particular learning and testing data, [ddalphaf.getErrorRatePart](#) to perform a benchmark study of the functional DD-classifier on particular data.

**Examples**

```
# load the fdata
df = dataf.growth()

stat <- ddalphaf.getErrorRateCV(dataf = df$dataf, labels = df$labels,
                               numchunks = 5,
                               adc.args = list(instance = "avr",
                                               numFcn = 2,
                                               numDer = 2))

cat("Classification error rate: ", stat$errors, ".\n", sep = "")
```

---

```
ddalphaf.getErrorRatePart
```

*Test Functional DD-Classifier*

---

**Description**

Performs a benchmark procedure by partitioning the given data. On each of times steps size observations are removed from the data, the functional DD-classifier is trained on these data and tested on the removed observations.

**Usage**

```
ddalphaf.getErrorRatePart(dataf, labels, size = 0.3, times = 10,
                          disc.type = c("LS", "comp"), ...)
```

**Arguments**

dataf	list containing lists (functions) of two vectors of equal length, named "args" and "vals": arguments sorted in ascending order and corresponding them values respectively
labels	list of output labels of the functional observations
size	the excluded sequences size. Either an integer between 1 and $n$ , or a fraction of data between 0 and 1.
times	the number of times the classifier is trained.
disc.type	type of the used discretization scheme. "LS" for <a href="#">ddalphaf.train</a> , "comp" for <a href="#">compclassf.train</a>
...	additional parameters passed to <a href="#">ddalphaf.train</a>

**Value**

errors	the part of incorrectly classified data (mean)
errors_sd	the standard deviation of errors
errors_vec	vector of errors
time	the mean training time
time_sd	the standard deviation of training time

**See Also**

[ddalphaf.train](#) to train the functional  $DD\alpha$ -classifier, [ddalphaf.classify](#) for classification using functional  $DD\alpha$ -classifier, [ddalphaf.test](#) to test the functional  $DD$ -classifier on particular learning and testing data, [ddalphaf.getErrorRateCV](#) to get error rate of the functional  $DD$ -classifier on particular data.

**Examples**

```
# load the fdata
df = dataf.growth()

stat <- ddalphaf.getErrorRatePart(dataf = df$dataf, labels = df$labels,
                                size = 0.3, times = 5,
                                adc.args = list(instance = "avr",
                                                numFcn = 2,
                                                numDer = 2))

cat("Classification error rate: ", stat$errors, ".\n", sep = "")
```

---

ddalphaf.test

*Test Functional DD-Classifier*


---

**Description**

Trains functional  $DD$ -classifier on the learning sequence of the data and tests it on the testing sequence.

**Usage**

```
ddalphaf.test(learn, learnlabels, test, testlabels, disc.type = c("LS", "comp"), ...)
```

**Arguments**

learn	list containing lists (functions) of two vectors of equal length, named "args" and "vals": arguments sorted in ascending order and corresponding them values respectively
learnlabels	list of output labels of the functional observations
test	the testing sequence. Has the same format as learn
disc.type	type of the used discretization scheme. "LS" for <a href="#">ddalphaf.train</a> , "comp" for <a href="#">compclassf.train</a>
testlabels	list of output labels of the functional observations
...	additional parameters passed to <a href="#">ddalphaf.train</a>

**Value**

error	the part of incorrectly classified data
correct	the number of correctly classified objects
incorrect	the number of incorrectly classified objects
total	the number of classified objects
ignored	the number of ignored objects (outside the convex hull of the learning data)
n	the number of objects in the testing sequence
time	training time

**See Also**

[ddalphaf.train](#) to train the functional  $DD\alpha$ -classifier, [ddalphaf.classify](#) for classification using functional  $DD\alpha$ -classifier, [ddalphaf.getErrorRateCV](#) and [ddalphaf.getErrorRatePart](#) to get error rate of the functional  $DD$ -classifier on particular data.

**Examples**

```
# load the fdata
df = dataf.growth()

samp = c(35:70)

ddalphaf.test(learn = df$dataf[-samp], learnlabels = df$labels[-samp],
              test = df$dataf[samp], testlabels = df$labels[samp],
              adc.args = list(instance = "avr",
                              numFcn = 2,
                              numDer = 2))
```

---

 ddalphaf.train                      *Functional DD-Classifier*


---

**Description**

Trains the functional DD-classifier

**Usage**

```
ddalphaf.train(dataf, labels, subset,
               adc.args = list(instance = "avr",
                               numFcn = -1,
                               numDer = -1),
               classifier.type = c("ddalpha", "maxdepth", "knnaff", "lda", "qda"),
               cv.complete = FALSE,
               maxNumIntervals = min(25, ceiling(length(dataf[[1]]$args)/2)),
               seed = 0,
               ...)
```

**Arguments**

dataf	list containing lists (functions) of two vectors of equal length, named "args" and "vals": arguments sorted in ascending order and corresponding them values respectively
labels	list of output labels of the functional observations
subset	an optional vector specifying a subset of observations to be used in training the classifier.
adc.args	Represents a function sample as a multidimensional (dimension="numFcn"+"numDer") one averaging (instance = "avr") or evaluating (instance = "val") for that each function and it derivative on "numFcn" (resp. "numDer") equal nonoverlapping covering intervals First two named "args" and "vals" are arguments sorted in ascending order and having same bounds for all functions and corresponding them values respectively <b>instance</b> type of discretizing the functions: "avr" - by averaging over intervals of the same length "val" - by taking values on equally-spaced grid <b>numFcn</b> number of function intervals <b>numDer</b> number of first-derivative intervals Set numFcn and numDer to -1 to apply cross-validation. Set adc.args to a list of "adc.args" objects to cross-validate only over these values.
classifier.type	the classifier which is used on the transformed space. The default value is 'ddalpha'.

`cv.complete`      T: apply complete cross-validation  
                       F: restrict cross-validation by Vapnik-Chervonenkis bound  
`maxNumIntervals`  
                       maximal number of intervals for cross-validation ( `max(numFcn + numDer) = maxNumIntervals` )  
`seed`  
                       the random seed. The default value `seed=0` makes no changes.  
`...`  
                       additional parameters, passed to the classifier, selected with parameter `classifier.type`.

### Details

The functional DD-classifier is fast nonparametric procedure for classifying functional data. It consists of a two-step transformation of the original data plus a classifier operating on a low-dimensional hypercube. The functional data are first mapped into a finite-dimensional location-slope space and then transformed by a multivariate depth function into the DD-plot, which is a subset of the unit hypercube. This transformation yields a new notion of depth for functional data. Three alternative depth functions are employed for this, as well as two rules for the final classification. The resulting classifier is cross-validated over a small range of parameters only, which is restricted by a Vapnik-Cervonenkis bound. The entire methodology does not involve smoothing techniques, is completely nonparametric and allows to achieve Bayes optimality under standard distributional settings. It is robust and efficiently computable.

### Value

Trained functional DD-classifier

### References

Mosler, K. and Mozharovskyi, P. (2017). Fast DD-classification of functional data. *Statistical Papers* **58** 1055–1089.

Mozharovskyi, P. (2015). *Contributions to Depth-based Classification and Computation of the Tukey Depth*. Verlag Dr. Kovac (Hamburg).

### See Also

[ddalphaf.classify](#) for classification using functional  $DD_\alpha$ -classifier, [compclassf.train](#) to train the functional componentwise classifier, [dataf.\\*](#) for functional data sets included in the package.

### Examples

```
## Not run:

## load the Growth dataset
dataf = dataf.growth()

learn = c(head(dataf$dataf, 49), tail(dataf$dataf, 34))
labels= c(head(dataf$labels, 49), tail(dataf$labels, 34))
test  = tail(head(dataf$dataf, 59), 10)  # elements 50:59. 5 girls, 5 boys
```

```

#cross-validate over the whole variants up to dimension 3
c1 = ddalphaf.train (learn, labels, classifier.type = "ddalpha", maxNumIntervals = 3)

classified1 = ddalphaf.classify(c1, test)

print(unlist(classified1))
print(c1$adc.args)

# cross-validate over these two variants
c2 = ddalphaf.train (learn, labels, classifier.type = "ddalpha",
                    adc.args = list(
                        list(instance = "avr",
                            numFcn = 1,
                            numDer = 2),
                        list(instance = "avr",
                            numFcn = 0,
                            numDer = 2)))

classified2 = ddalphaf.classify(c2, test)

print(unlist(classified2))
print(c2$adc.args)

## End(Not run)

```

---

depth.
*Calculate Depth*


---

### Description

Calculates the depth of points w.r.t. a multivariate data set.

The detailed descriptions are found in the corresponding topics.

### Usage

```
depth.(x, data, notion, ...)
```

```
## beta-skeleton depth
```

```
# depth.betaSkeleton(x, data, beta = 2, distance = "Lp", Lp.p = 2,
```

```
# mah.estimate = "moment", mah.parMcd = 0.75)
```

```
## Tukey depth
```

```
# depth.halfspace(x, data, exact, method, num.directions = 1000, seed = 0)
```

```
## L2-depth
```

```
# depth.L2(x, data, mah.estimate = "moment", mah.parMcd = 0.75)
```

```
## Mahalanobis depth
```

```

# depth.Mahalanobis(x, data, mah.estimate = "moment", mah.parMcd = 0.75)

## projection depth
# depth.projection(x, data, method = "random", num.directions = 1000)

## simplicial depth
# depth.simplicial(x, data, exact = F, k = 0.05, seed = 0)

## simplicial volume depth
# depth.simplicialVolume(x, data, exact = F, k = 0.05, seed = 0)

## spatial depth
# depth.spatial(x, data)

## zonoid depth
# depth.zonoid(x, data)

## potential
# depth.potential (x, data, pretransform = "1Mom",
#                 kernel = "GKernel", kernel.bandwidth = NULL, mah.parMcd = 0.75)

## convex hull peeling depth
# depth.qhpeeling(x, data)

```

### Arguments

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
notion	The name of the depth notion (shall also work with a user-defined depth function named "depth.<name>").
...	Additional parameters passed to the depth functions.

### Value

Numerical vector of depths, one for each row in x; or one depth value if x is a numerical vector.

### See Also

[depth.betaSkeleton](#)  
[depth.halfspace](#)  
[depth.L2](#)  
[depth.Mahalanobis](#)  
[depth.projection](#)

```

depth.simplicial
depth.simplicialVolume
depth.spatial
depth.zonoid
depth.potential
depth.qhpeeling
depth.graph for building the depth surfaces of the two dimensional data.

```

### Examples

```

# 5-dimensional normal distribution
data <- mvrnorm(1000, rep(0, 5),
               matrix(c(1, 0, 0, 0, 0,
                        0, 2, 0, 0, 0,
                        0, 0, 3, 0, 0,
                        0, 0, 0, 2, 0,
                        0, 0, 0, 0, 1),
                      nrow = 5))
x <- mvrnorm(10, rep(1, 5),
             matrix(c(1, 0, 0, 0, 0,
                     0, 1, 0, 0, 0,
                     0, 0, 1, 0, 0,
                     0, 0, 0, 1, 0,
                     0, 0, 0, 0, 1),
                   nrow = 5))

depths <- depth.(x, data, notion = "zonoid")
cat("Depths: ", depths, "\n")

```

---

depth.betaSkeleton	<i>Calculate Beta-Skeleton Depth</i>
--------------------	--------------------------------------

---

### Description

Calculates the beta-skeleton depth of points w.r.t. a multivariate data set.

### Usage

```

depth.betaSkeleton(x, data, beta = 2, distance = "Lp", Lp.p = 2,
                  mah.estimate = "moment", mah.parMcd = 0.75)

```

### Arguments

**x** Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a  $d$ -variate point. Should have the same dimension as data.

data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
beta	The parameter defining the positioning of the balls' centers, see Yang and Modarres (2017) for details. By default (together with other arguments) equals 2, which corresponds to the lens depth, see Liu and Modarres (2011).
distance	A character string defining the distance to be used for determining inclusion of a point into the lens (influence region), see Yang and Modarres (2017) for details. Possibilities are "Lp" for the Lp-metric (default) or "Mahalanobis" for the Mahalanobis distance adjustment.
Lp.p	A non-negative number defining the distance's power equal 2 by default (Euclidean distance); is used only when distance = "Lp".
mah.estimate	A character string specifying which estimates to use when calculating sample covariance matrix; can be "none", "moment" or "MCD", determining whether traditional moment or Minimum Covariance Determinant (MCD) (see <a href="#">covMcd</a> ) estimates for mean and covariance are used. By default "moment" is used. Is used only when distance = "Mahalanobis".
mah.parMcd	The value of the argument alpha for the function <a href="#">covMcd</a> ; is used when distance = "Mahalanobis" and mah.estimate = "MCD".

### Details

Calculates the beta-skeleton depth, see Yang and Modarres (2017). Its particular case, lens depth, see Liu and Modarres (2011), is obtained when beta = 2, distance = "Lp" and Lp.p = 2 (default settings). For the example of the lens depth, the depth of an observation  $x$  is calculated as the portion of lens containing  $x$ , with lens being an intersection of two closed balls centered at two sample's points each having radius equal to the distance between these two points.

### Value

Numerical vector of depths, one for each row in  $x$ ; or one depth value if  $x$  is a numerical vector.

### References

- Liu, Z. and Modarres, R. (2011). Lens data depth and median. *Journal of Nonparametric Statistics* **23**(4) 1063–1074.
- Yang, M. and Modarres, R. (2017).  $\beta$ -skeleton depth functions and medians. *Communications in Statistics - Theory and Methods* to appear.

### See Also

- [depth.halfspace](#) for calculation of the Tukey depth.
- [depth.Mahalanobis](#) for calculation of Mahalanobis depth.
- [depth.projection](#) for calculation of projection depth.
- [depth.simplicial](#) for calculation of simplicial depth.
- [depth.simplicialVolume](#) for calculation of simplicial volume depth.
- [depth.spatial](#) for calculation of spatial depth.

[depth.zonoid](#) for calculation of zonoid depth.

[depth.potential](#) for calculation of data potential.

### Examples

```
# 5-dimensional normal distribution
data <- mvrnorm(1000, rep(0, 5),
               matrix(c(1, 0, 0, 0, 0,
                        0, 2, 0, 0, 0,
                        0, 0, 3, 0, 0,
                        0, 0, 0, 2, 0,
                        0, 0, 0, 0, 1),
                      nrow = 5))
x <- mvrnorm(10, rep(1, 5),
             matrix(c(1, 0, 0, 0, 0,
                     0, 1, 0, 0, 0,
                     0, 0, 1, 0, 0,
                     0, 0, 0, 1, 0,
                     0, 0, 0, 0, 1),
                   nrow = 5))

depths <- depth.betaSkeleton(x, data)
cat("Depths:", depths, "\n")
```

---

depth.contours

*Depth Contours*

---

### Description

Builds the data depth contours for 2-dimensional data.

### Usage

```
depth.contours(data, depth,
               main = "", xlab="", ylab = "",
               drawplot = T, frequency=100, levels = 10,
               col = "red",
               ...)
```

### Arguments

data	2-dimensional numeric data frame or matrix
depth	the name of the depth function. The list of the supported depths and described in the topic <a href="#">depth..</a>
main	an overall title for the plot: see <a href="#">title</a>
xlab, ylab	labels of the axes
drawplot	if set to false, the contours are built on the existing plot.

frequency	number of points on each direction, x and y. Impacts the smoothness of the contours.
levels	numeric vector of levels at which to draw contour lines. If the vector contains only ONE element, the levels are generated automatically as <code>seq(0,max(depth),length.out = levels)</code> .
col	color, used to draw points and contours
...	additional parameters passed to the depth functions and to <a href="#">plot</a>

**See Also**

[depth.](#), [depth.contours.ddalpha](#), [depth.graph](#).

**Examples**

```
## Not run:

par(mfrow = c(2,2))
data(hemophilia)

depth.contours(hemophilia[,1:2], depth = "none", main = "data")

for (depth in c("zonoid", "Mahalanobis", "projection", "spatial")){
  depth.contours(hemophilia[,1:2], depth = depth, main = depth)
}

for (depth in c("halfspace", "simplicial", "simplicialVolume")){
  depth.contours(hemophilia[,1:2], depth = depth, main = depth, exact = T)
}

## End(Not run)
```

---

`depth.contours.ddalpha`

*Depth Contours*

---

**Description**

Builds the data depth contours for multiclass 2-dimensional data using the trained classifier. Also accessible from [plot.ddalpha](#).

**Usage**

```
depth.contours.ddalpha(ddalpha,
  main = "", xlab="", ylab = "",
  drawplot = T, frequency=100, levels = 10, drawsep = T, ...)
```

**Arguments**

ddalpha	DD $\alpha$ -classifier (obtained by <code>ddalpha.train</code> ).
main	an overall title for the plot: see <code>title</code>
xlab, ylab	labels of the axes
drawplot	if set to false, the contours are built on the existing plot.
frequency	number of points on each direction, x and y. Impacts the smoothness of the contours.
levels	numeric vector of levels at which to draw contour lines. If the vector contains only ONE element, the levels are generated automatically as <code>seq(0, max(depth), length.out = levels)</code> .
drawsep	draws the separation on the DD-plot (currently for 2 classes and not for knn)
...	additional parameters passed to the depth functions and to <code>plot</code>

**See Also**

`depth.`, `depth.contours`, `depth.graph`.

**Examples**

```
## Not run:

par(mfrow = c(2,2))
data(hemophilia)

ddalpha = ddalpha.train(hemophilia, depth = "none")
depth.contours.ddalpha(ddalpha, main = "data")

for (depth in c("zonoid", "Mahalanobis", "projection", "spatial")){
  ddalpha = ddalpha.train(hemophilia, depth = depth)
  depth.contours.ddalpha(ddalpha, main = depth)
}

for (depth in c("halfspace", "simplicial", "simplicialVolume")){
  ddalpha = ddalpha.train(hemophilia, depth = depth, exact = T)
  depth.contours.ddalpha(ddalpha, main = depth)
}

## End(Not run)
```

depth.graph

*Depth Graph***Description**

Builds the data depth graphs for 2-dimensional data. The graph is built using [persp](#).

**Usage**

```
depth.graph(data,
  depth_f = c("halfspace", "Mahalanobis", "projection", "simplicial",
             "simplicialVolume", "spatial", "zonoid", "none"),
  apoint = NULL,
  main = depth_f,
  xlim = c(min(data[, 1]), max(data[, 1])),
  ylim = c(min(data[, 2]), max(data[, 2])),
  zlim = c(0, max(z)),
  xnum = 250,
  ynum = 250,
  theta=15, phi=60,
  bold = F,
  ...)
```

**Arguments**

data	2-dimensional numeric data frame or matrix
depth_f	the name of the depth function. The list of the supported depths and described in the topic <a href="#">depth..</a>
apoint	a 2-dimensional point which is shown in black color.
main	an overall title for the plot: see <a href="#">title</a>
xlim, ylim, zlim	numeric vectors of length 2, giving the x, y and z coordinates ranges: see <a href="#">plot.window</a>
xnum, ynum	number of points on each direction, x and y. Impacts the smoothness of the surface.
theta, phi	rotation angles
bold	draws bold points
...	additional parameters passed to <a href="#">persp</a>

**See Also**

[depth.](#)  
[persp](#)

**Examples**

```
## Not run:

par(mfrow = c(2,3), mar = c(0,0,0,0), mai = c(0,0,0.2,0))
data(hemophilia)
depth.graph(hemophilia, "none", xnum = 100, ynum = 100)
depth.graph(hemophilia, "Mahalanobis", xnum = 100, ynum = 100)
depth.graph(hemophilia, "halfspace", xnum = 100, ynum = 100)
depth.graph(hemophilia, "projection", xnum = 100, ynum = 100)
depth.graph(hemophilia, "zonoid", xnum = 100, ynum = 100)
depth.graph(hemophilia, "spatial", xnum = 100, ynum = 100)

## End(Not run)
```

---

depth.halfspace      *Calculate Halfspace Depth*

---

**Description**

Calculates the exact or random Tukey (=halfspace, location) depth (Tukey, 1975) of points w.r.t. a multivariate data set.

**Usage**

```
depth.halfspace(x, data, exact, method, num.directions = 1000, seed = 0)
```

**Arguments**

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
exact	The type of the used method. The default is exact=F, which leads to approximate computation of the Tukey depth. For exact=F, method="Sunif.1D" is used by default. If exact=T, the Tukey depth is computed exactly, with method="recursive" by default.
method	For exact=F, if method="Sunif.1D" (by default), the Tukey depth is computed approximately by being minimized over univariate projections (see Details below). For exact=T, the Tukey depth is calculated as the minimum over all combinations of $k$ points from data (see Details below). In this case parameter method specifies $k$ , with possible values 1 for method="recursive" (by default), $d - 2$ for method="plane", $d - 1$ for method="line". The name of the method may be given as well as just parameter exact, in which case the default method will be used.

num.directions Number of random directions to be generated (for method="Sunif.1D"). The algorithmic complexity is linear in the number of observations in data, given the number of directions.

seed The random seed. The default value seed=0 makes no changes (for method="Sunif.1D").

### Details

For exact=F, if method="Sunif.1D", the Tukey depth is computed approximately using the random Tukey depth method proposed by Cuesta-Albertos and Nieto-Reyes (2008). Here the depth is determined as the minimum univariate Tukey depth of the - on lines in several directions - projected data. The directions are distributed uniformly on the  $(d - 1)$ -sphere; the same direction set is used for all points.

For exact=T, the Tukey depth is computed exactly as the minimum of the sum of the depths in two orthogonal complementary affine subspaces, which dimensions add to  $d$ : one of the subspaces (combinatorial) is the  $k$ -dimensional hyperplane through (a point from)  $x$  and  $k$  points from data, another one is its orthogonal complement (see Dyckerhoff and Mozharovskyi, 2016 for the detailed description of the algorithmic framework). The algorithm then minimizes the depth over all combinations of  $k$  points, in which the depth in the orthogonal complements is computed using an exact algorithm. In this case, parameter method specifies the dimensionality  $k$  of the combinatorial space. The implemented (reasonable) algorithms (and corresponding names) are:  $k = 1$  (or method="recursive"),  $k = d - 2$  (or method="plane"), and  $k = d - 1$  (or method="line").

### Value

Numerical vector of depths, one for each row in  $x$ ; or one depth value if  $x$  is a numerical vector.

### References

- Cuesta-Albertos, J.A. and Nieto-Reyes, A. (2008). The random Tukey depth. *Computational Statistics and Data Analysis* **52** 4979–4988.
- Dyckerhoff, R. and Mozharovskyi, P. (2016). Exact computation of the halfspace depth. *Computational Statistics and Data Analysis* **98** 19–30.
- Rousseeuw, P.J. and Ruts, I. (1996). Algorithm AS 307: Bivariate location depth. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **45** 516–526.
- Tukey, J.W. (1974). Mathematics and the picturing of data. In: *Proceeding of the International Congress of Mathematicians*, Vancouver, 523–531.

### See Also

- [depth.Mahalanobis](#) for calculation of Mahalanobis depth.
- [depth.projection](#) for calculation of projection depth.
- [depth.simplicial](#) for calculation of simplicial depth.
- [depth.simplicialVolume](#) for calculation of simplicial volume depth.
- [depth.spatial](#) for calculation of spatial depth.
- [depth.zonoid](#) for calculation of zonoid depth.
- [depth.potential](#) for calculation of data potential.

**Examples**

```

# 3-dimensional normal distribution
data <- mvrnorm(200, rep(0, 3),
              matrix(c(1, 0, 0,
                      0, 2, 0,
                      0, 0, 1),
                    nrow = 3))
x <- mvrnorm(10, rep(1, 3),
            matrix(c(1, 0, 0,
                    0, 1, 0,
                    0, 0, 1),
                  nrow = 3))

# default - random Tukey depth
depths <- depth.halfspace(x, data)
cat("Depths: ", depths, "\n")

# default exact method - "recursive"
depths <- depth.halfspace(x, data, exact = TRUE)
cat("Depths: ", depths, "\n")

# method "line"
depths <- depth.halfspace(x, data, method = "line")
cat("Depths: ", depths, "\n")

```

---

depth.L2

*Calculate L2-Depth*


---

**Description**

Calculates the L2-depth of points w.r.t. a multivariate data set.

**Usage**

```
depth.L2(x, data, mah.estimate = "moment", mah.parMcd = 0.75)
```

**Arguments**

<code>x</code>	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as <code>data</code> .
<code>data</code>	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
<code>mah.estimate</code>	is a character string specifying which estimates to use when calculating sample covariance matrix; can be "none", "moment" or "MCD", determining whether traditional moment or Minimum Covariance Determinant (MCD) (see <a href="#">covMcd</a> ) estimates for mean and covariance are used. By default "moment" is used. With "none" the non-affine invariant version of the L2-depth is calculated

`mah.parMcd` is the value of the argument `alpha` for the function `covMcd`; is used when `mah.estimate = "MCD"`.

### Details

Calculates L2-depth (Mosler, 2013). L2-depth is based on the outlyingness distance calculated as the average L2-distance from (a row of) `x` to each point in data.

### Value

Numerical vector of depths, one for each row in `x`; or one depth value if `x` is a numerical vector.

### References

Mosler, K. (2013). Depth statistics. In: Becker, C., Fried, R. and Kuhnt, S. (eds), *Robustness and Complex Data Structures: Festschrift in Honour of Ursula Gather*, Springer-Verlag (Berlin, Heidelberg), 17–34.

### See Also

[depth.halfspace](#) for calculation of the Tukey depth.  
[depth.Mahalanobis](#) for calculation of Mahalanobis depth.  
[depth.projection](#) for calculation of projection depth.  
[depth.qhpeeling](#) for calculation of convex hull peeling depth.  
[depth.simplicial](#) for calculation of simplicial depth.  
[depth.simplicialVolume](#) for calculation of simplicial volume depth.  
[depth.spatial](#) for calculation of spatial depth.  
[depth.potential](#) for calculation of data potential.  
[depth.zonoid](#) for calculation of zonoid depth.

### Examples

```
# 5-dimensional normal distribution
data <- mvrnorm(1000, rep(0, 5),
               matrix(c(1, 0, 0, 0, 0,
                       0, 2, 0, 0, 0,
                       0, 0, 3, 0, 0,
                       0, 0, 0, 2, 0,
                       0, 0, 0, 0, 1),
                     nrow = 5))
x <- mvrnorm(10, rep(1, 5),
             matrix(c(1, 0, 0, 0, 0,
                     0, 1, 0, 0, 0,
                     0, 0, 1, 0, 0,
                     0, 0, 0, 1, 0,
                     0, 0, 0, 0, 1),
                   nrow = 5))
```

```
depths <- depth.spatial(x, data)
cat("Depths:", depths, "\n")
```

---

depth.Mahalanobis      *Calculate Mahalanobis Depth*

---

### Description

Calculates the Mahalanobis depth of points w.r.t. a multivariate data set.

### Usage

```
depth.Mahalanobis(x, data, mah.estimate = "moment", mah.parMcd = 0.75)
```

### Arguments

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
mah.estimate	is a character string specifying which estimates to use when calculating the Mahalanobis depth; can be "moment" or "MCD", determining whether traditional moment or Minimum Covariance Determinant (MCD) (see <a href="#">covMcd</a> ) estimates for mean and covariance are used. By default "moment" is used.
mah.parMcd	is the value of the argument alpha for the function <a href="#">covMcd</a> ; is used when mah.estimate = "MCD".

### Details

Calculates Mahalanobis depth. Mahalanobis depth is based on an outlyingness measure (Zuo & Serfling, 2000), *viz.* the Mahalanobis distance between the given point and the center of the data (Mahalanobis, 1936).

*Moment estimates* may be used i.e. traditional *mean* and *covariance matrix*, the corresponding depth may be sensitive to outliers. A more robust depth is obtained with *minimum volume ellipsoid* (MVE) or *minimum covariance determinant* (MCD) estimators, see Rousseeuw & Leroy (1987) and Lopuhaa & Rousseeuw (1991).

### Value

Numerical vector of depths, one for each row in x; or one depth value if x is a numerical vector.

## References

- Mahalanobis, P. (1936). On the generalized distance in statistics. *Proceedings of the National Academy India* **12** 49–55.
- Liu, R.Y. (1992). Data depth and multivariate rank tests. In: Dodge, Y. (ed.), *LI-Statistics and Related Methods*, North-Holland (Amsterdam), 279–294.
- Lopuhaa, H.P. and Rousseeuw, P.J. (1991). Breakdown points of affine equivariant estimators of multivariate location and covariance matrices. *The Annals of Statistics* **19** 229–248.
- Rousseeuw, P.J. and Leroy, A.M. (1987). *Robust Regression and Outlier Detection*. John Wiley & Sons (New York).
- Zuo, Y.J. and Serfling, R. (2000). General notions of statistical depth function. *The Annals of Statistics* **28** 461–482.

## See Also

- [depth.halfspace](#) for calculation of the Tukey depth.
- [depth.projection](#) for calculation of projection depth.
- [depth.simplicial](#) for calculation of simplicial depth.
- [depth.simplicialVolume](#) for calculation of simplicial volume depth.
- [depth.spatial](#) for calculation of spatial depth.
- [depth.zonoid](#) for calculation of zonoid depth.
- [depth.potential](#) for calculation of data potential.

## Examples

```
# 5-dimensional normal distribution
data <- mvrnorm(1000, rep(0, 5),
               matrix(c(1, 0, 0, 0, 0,
                       0, 2, 0, 0, 0,
                       0, 0, 3, 0, 0,
                       0, 0, 0, 2, 0,
                       0, 0, 0, 0, 1),
                     nrow = 5))
x <- mvrnorm(10, rep(1, 5),
             matrix(c(1, 0, 0, 0, 0,
                     0, 1, 0, 0, 0,
                     0, 0, 1, 0, 0,
                     0, 0, 0, 1, 0,
                     0, 0, 0, 0, 1),
                   nrow = 5))

depths <- depth.Mahalanobis(x, data)
cat("Depths moment: ", depths, "\n")
depths <- depth.Mahalanobis(x, data, mah.estimate = "MCD", mah.parMcd = 0.75)
cat("Depths MCD: ", depths, "\n")
```

---

depth.potential      *Calculate Potential of the Data*

---

### Description

Calculate the potential of the points w.r.t. a multivariate data set. The potential is the kernel-estimated density multiplied by the prior probability of a class. Different from the data depths, a density estimate measures at a given point how much mass is located around it.

### Usage

```
depth.potential(x, data, pretransform = "1Mom",
               kernel = "GKernel", kernel.bandwidth = NULL, mah.parMcd = 0.75)
```

### Arguments

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
pretransform	The method of data scaling. NULL to use the original data, 1Mom or NMom for scaling using data moments, 1MCD or NMCD for scaling using robust data moments (Minimum Covariance Determinant (MCD)).
kernel	"EDKernel" for the kernel of type $1/(1+kernel.bandwidth*EuclidianDistance2(x, y))$ , "GKernel" [default and recommended] for the simple Gaussian kernel, "EKernel" exponential kernel: $\exp(-kernel.bandwidth*EuclidianDistance(x, y))$ , "VarGKernel" variable Gaussian kernel, where kernel.bandwidth is proportional to the depth.zonoid of a point.
kernel.bandwidth	the single bandwidth parameter of the kernel. If NULL - the Scott's rule of thumb is used.
mah.parMcd	is the value of the argument alpha for the function <code>covMcd</code> ; is used when pretransform = "*MCD".

### Details

The potential is the kernel-estimated density multiplied by the prior probability of a class. The kernel bandwidth matrix is decomposed into two parts, one of which describes the form of the data, and the other the width of the kernel. Then the first part is used to transform the data using the moments, while the second is employed as a parameter of the kernel and tuned to achieve the best separation. For details see Pokotylo and Mosler (2015).

**Value**

Numerical vector of potentials, one for each row in `x`; or one potential value if `x` is a numerical vector.

**References**

Aizerman, M.A., Braverman, E.M., and Rozonoer, L.I. (1970). *The Method of Potential Functions in the Theory of Machine Learning*. Nauka (Moscow).

Pokotylo, O. and Mosler, K. (2015). Classification with the pot-pot plot. *Mimeo*.

**See Also**

[depth.halfspace](#) for calculation of the Tukey depth.

[depth.Mahalanobis](#) for calculation of Mahalanobis depth.

[depth.projection](#) for calculation of projection depth.

[depth.simplicial](#) for calculation of simplicial depth.

[depth.simplicialVolume](#) for calculation of simplicial volume depth.

[depth.spatial](#) for calculation of spatial depth.

[depth.zonoid](#) for calculation of zonoid depth.

**Examples**

```
# 3-dimensional normal distribution
data <- mvrnorm(200, rep(0, 3),
               matrix(c(1, 0, 0,
                        0, 2, 0,
                        0, 0, 1),
                       nrow = 3))
x <- mvrnorm(10, rep(1, 3),
             matrix(c(1, 0, 0,
                     0, 1, 0,
                     0, 0, 1),
                   nrow = 3))

# potential with rule of thumb bandwidth
pot <- depth.potential(x, data)
cat("Potentials: ", pot, "\n")

# potential with bandwidth = 0.1
pot <- depth.potential(x, data, kernel.bandwidth = 0.1)
cat("Potentials: ", pot, "\n")

# potential with robust MCD scaling
pot <- depth.potential(x, data, kernel.bandwidth = 0.1,
                      pretransform = "NMCD", mah.parMcd = 0.6)
cat("Potentials: ", pot, "\n")
```

---

depth.projection      *Calculate Projection Depth*

---

### Description

Calculates the projection depth of points w.r.t. a multivariate data set.

### Usage

```
depth.projection(x, data, method = "random", num.directions = 1000, seed = 0)
```

### Arguments

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
method	to be used in calculations. "random" Here the depth is determined as the minimum univariate depth of the data projected on lines in several directions. The directions are distributed uniformly on the $(d - 1)$ -sphere; the same direction set is used for all points. "linearize" The Nelder-Mead method for function minimization, taken from Olsson, Journal of Quality Technology, 1974, 6, 56.
num.directions	Number of random directions to be generated for method = "random". With the growth of $n$ the complexity grows linearly for the same number of directions.
seed	the random seed. The default value seed=0 makes no changes.

### Details

Calculates projection depth. Projection depth, similar to Mahalanobis depth, is based on a measure of outlyingness, used by Stahel (1981) and Donoho (1982), and has been first formulated by Liu (1992). The worst case outlyingness is obtained by maximizing an outlyingness measure over all univariate projections. In practice most often *median*, and *median absolute deviation from the median* (MAD), are used as they are robust measures.

### Value

Numerical vector of depths, one for each row in  $x$ ; or one depth value if  $x$  is a numerical vector.

### Author(s)

R-codes for the "linearize" method were written by Subhajit Dutta.

## References

- Donoho, D.L. (1982). *Breakdown properties of multivariate location estimators*. Ph.D. qualifying paper. Department of Statistics, Harvard University.
- Liu, R.Y. (1992). Data depth and multivariate rank tests. In: Dodge, Y. (ed.), *L1-Statistics and Related Methods*, North-Holland (Amsterdam), 279–294.
- Liu, X. and Zuo, Y. (2014). Computing projection depth and its associated estimators. *Statistics and Computing* **24** 51–63.
- Stahel, W.A. (1981). *Robust estimation: infinitesimal optimality and covariance matrix estimators*. Ph.D. thesis (in German). Eidgenössische Technische Hochschule Zurich.
- Zuo, Y.J. and Lai, S.Y. (2011). Exact computation of bivariate projection depth and the Stahel-Donoho estimator. *Computational Statistics and Data Analysis* **55** 1173–1179.

## See Also

- [depth.halfspace](#) for calculation of the Tukey depth.
- [depth.Mahalanobis](#) for calculation of Mahalanobis depth.
- [depth.simplicial](#) for calculation of simplicial depth.
- [depth.simplicialVolume](#) for calculation of simplicial volume depth.
- [depth.spatial](#) for calculation of spatial depth.
- [depth.zonoid](#) for calculation of zonoid depth.
- [depth.potential](#) for calculation of data potential.

## Examples

```
# 5-dimensional normal distribution
data <- mvrnorm(100, rep(0, 5),
               matrix(c(1, 0, 0, 0, 0,
                       0, 2, 0, 0, 0,
                       0, 0, 3, 0, 0,
                       0, 0, 0, 2, 0,
                       0, 0, 0, 0, 1),
                     nrow = 5))
x <- mvrnorm(10, rep(1, 5),
             matrix(c(1, 0, 0, 0, 0,
                     0, 1, 0, 0, 0,
                     0, 0, 1, 0, 0,
                     0, 0, 0, 1, 0,
                     0, 0, 0, 0, 1),
                   nrow = 5))

depths <- depth.projection(x, data, method = "random", num.directions = 1000)
cat("Depths random: ", depths, "\n")
depths <- depth.projection(x, data, method = "linearize")
cat("Depths linearize: ", depths, "\n")
```

---

depth.qhpeeling	<i>Calculate Convex Hull Peeling Depth</i>
-----------------	--

---

### Description

Calculates the convex hull peeling depth of points w.r.t. a multivariate data set.

### Usage

```
depth.qhpeeling(x, data)
```

### Arguments

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.

### Details

Calculates the convex hull peeling depth (Eddy, 1982; see also Cascos, 2009).

### Value

Numerical vector of depths, one for each row in `x`; or one depth value if `x` is a numerical vector. Each depth value equals the number of the convex hulls to be peeled from `data` so that (the corresponding row of) `x` is not contained in the convex hull of the rest of the data; the depths are normalized by the number of points in `data`.

### References

Eddy, W.F. (1982). Convex hull peeling. In: Caussinus, H., Ettinger, P. and Tomassone, R. (eds), *COMPSTAT 1982. Proceedings in computational statistics*, Physica-Verlag (Vienna), 42–47.

Cascos, I. (2009). Data depth: multivariate statistics and geometry. In: Kendall, W.S. and Molchanov, I. (eds) *New Perspectives in Stochastic Geometry*, Clarendon/Oxford University Press (Oxford).

### See Also

[depth.halfspace](#) for calculation of the Tukey depth.

[depth.L2](#) for calculation of L2-depth.

[depth.Mahalanobis](#) for calculation of Mahalanobis depth.

[depth.projection](#) for calculation of projection depth.

[depth.simplicial](#) for calculation of simplicial depth.

[depth.simplicialVolume](#) for calculation of simplicial volume depth.

`depth.spatial` for calculation of spatial depth.

`depth.potential` for calculation of data potential.

`depth.zonoid` for calculation of zonoid depth.

### Examples

```
# Mixture of 3-variate normal distributions
data <- mvrnorm(25, rep(0, 3), diag(3))
x <- rbind(mvrnorm(10, rep(1, 3), diag(3)), data)
depths <- depth.qhpeeling(x, data)
cat("Depths:", depths, "\n")
```

---

depth.sample	<i>Fast Depth Computation for Univariate and Bivariate Random Samples</i>
--------------	---

---

### Description

Faster implementation of the halfspace and the simplicial depth. Computes the depth of a whole random sample of a univariate or a bivariate data in one run.

### Usage

```
depth.sample(A, B)
```

### Arguments

A	Univariate or bivariate points whose depth is computed, represented by a matrix of size $m \times 2$ . $m$ stands for the number of points, $d$ is 1 for univariate and 2 for bivariate data.
B	Random sample points with respect to which the depth of A is computed. B is represented by a matrix of size $n \times 2$ , where $n$ is the sample size.

### Details

The function returns vectors of sample halfspace and simplicial depth values.

### Value

Vector of length  $m$  of depth halfspace depth values is returned.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**See Also**

[depth.halfspace](#)  
[depth.simplicial](#)

**Examples**

```
n = 100
m = 150
A = matrix(rnorm(2*n), ncol=2)
B = matrix(rnorm(2*m), ncol=2)
depth.sample(A,B)
system.time(D1<-depth.halfspace(A,B))
system.time(D2<-depth.sample(A,B))
max(D1-D2$Half)

A = rnorm(100)
B = rnorm(150)
depth.sample(A,B)
# depth.halfspace(matrix(A,ncol=1),matrix(B,ncol=1))
```

---

depth.simplicial      *Calculate Simplicial Depth*

---

**Description**

Calculates the simplicial depth of points w.r.t. a multivariate data set.

**Usage**

```
depth.simplicial(x, data, exact = F, k = 0.05, seed = 0)
```

**Arguments**

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
exact	exact=F (by default) implies the approximative algorithm, considering $k$ simplices, exact=T implies the exact algorithm.
k	Number ( $k > 1$ ) or portion (if $0 < k < 1$ ) of simplices that are considered if exact=F. If $k > 1$ , then the algorithmic complexity is polynomial in $d$ but is independent of the number of observations in data, given $k$ . If $0 < k < 1$ , then the algorithmic complexity is exponential in the number of observations in data, but the calculation precision stays approximately the same.
seed	the random seed. The default value seed=0 makes no changes.

**Details**

Calculates simplicial depth. Simplicial depth is counted as a probability that a point lies in a simplex, built on  $d + 1$  data points.

**Value**

Numerical vector of depths, one for each row in  $x$ ; or one depth value if  $x$  is a numerical vector.

**References**

Chaudhuri, P. (1996). On a geometric notion of quantiles for multivariate data. *Journal of the American Statistical Association* **91** 862–872.

Liu, R. Y. (1990). On a notion of data depth based on random simplices. *The Annals of Statistics* **18** 405–414.

Rousseeuw, P.J. and Ruts, I. (1996). Algorithm AS 307: Bivariate location depth. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **45** 516–526.

**See Also**

[depth.halfspace](#) for calculation of the Tukey depth.

[depth.Mahalanobis](#) for calculation of Mahalanobis depth.

[depth.projection](#) for calculation of projection depth.

[depth.simplicialVolume](#) for calculation of simplicial volume depth.

[depth.spatial](#) for calculation of spatial depth.

[depth.zonoid](#) for calculation of zonoid depth.

[depth.potential](#) for calculation of data potential.

**Examples**

```
# 3-dimensional normal distribution
data <- mvrnorm(20, rep(0, 3),
              matrix(c(1, 0, 0,
                      0, 2, 0,
                      0, 0, 1),
                    nrow = 3))
x <- mvrnorm(10, rep(1, 3),
            matrix(c(1, 0, 0,
                    0, 1, 0,
                    0, 0, 1),
                  nrow = 3))

#exact
depths <- depth.simplicial(x, data, exact = TRUE)
cat("Depths: ", depths, "\n")

#approximative
depths <- depth.simplicial(x, data, exact = FALSE, k = 0.2)
cat("Depths: ", depths, "\n")
```

---

 depth.simplicialVolume

*Calculate Simplicial Volume Depth*


---

### Description

Calculates the simplicial volume depth of points w.r.t. a multivariate data set.

### Usage

```
depth.simplicialVolume(x, data, exact = F, k = 0.05, mah.estimate = "moment",
  mah.parMcd = 0.75, seed = 0)
```

### Arguments

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
exact	exact=F (by default) implies the approximative algorithm, considering $k$ simplices, exact=T implies the exact algorithm.
k	Number ( $k > 1$ ) or portion (if $0 < k < 1$ ) of simplices that are considered if exact=F. If $k > 1$ , then the algorithmic complexity is polynomial in $d$ but is independent of the number of observations in data, given $k$ . If $0 < k < 1$ , then the algorithmic complexity is exponential in the number of observations in data, but the calculation precision stays approximately the same.
mah.estimate	A character string specifying affine-invariance adjustment; can be "none", "moment" or "MCD", determining whether no affine-invariance adjustment or moment or Minimum Covariance Determinant (MCD) (see <a href="#">covMcd</a> ) estimates of the covariance are used. By default "moment" is used.
mah.parMcd	The value of the argument alpha for the function <a href="#">covMcd</a> ; is used when mah.estimate = "MCD".
seed	The random seed. The default value seed=0 makes no changes.

### Details

Calculates Oja depth (also: Simplicial volume depth). At first the Oja outlyingness function  $O(x, data)$  is calculated as the average of the volumes of simplices built on  $d$  data points and the measurement point  $x$  (Oja, 1983).

Zuo and Serfling (2000) proposed Oja depth based on the Oja outlyingness function as  $1/(1 + O(x, data)/S)$ , where  $S$  is a square root of the determinant of  $cov(data)$ , which makes the depth function affine-invariant.

**Value**

Numerical vector of depths, one for each row in  $x$ ; or one depth value if  $x$  is a numerical vector.

**References**

Oja, H. (1983). Descriptive statistics for multivariate distributions. *Statistics & Probability Letters* **1** 327–332.

Zuo, Y.J. and Serfling, R. (2000). General notions of statistical depth function. *The Annals of Statistics* **28** 461–482.

**See Also**

[depth.halfspace](#) for calculation of the Tukey depth.

[depth.Mahalanobis](#) for calculation of Mahalanobis depth.

[depth.projection](#) for calculation of projection depth.

[depth.simplicial](#) for calculation of simplicial depth.

[depth.spatial](#) for calculation of spatial depth.

[depth.zonoid](#) for calculation of zonoid depth.

[depth.potential](#) for calculation of data potential.

**Examples**

```
# 3-dimensional normal distribution
data <- mvrnorm(20, rep(0, 3),
               matrix(c(1, 0, 0,
                        0, 2, 0,
                        0, 0, 1),
                       nrow = 3))
x <- mvrnorm(10, rep(1, 3),
             matrix(c(1, 0, 0,
                     0, 1, 0,
                     0, 0, 1),
                   nrow = 3))

#exact
depths <- depth.simplicialVolume(x, data, exact = TRUE)
cat("Depths: ", depths, "\n")

#approximative
depths <- depth.simplicialVolume(x, data, exact = FALSE, k = 0.2)
cat("Depths: ", depths, "\n")
```

---

depth.space.                      *Calculate Depth Space using the Given Depth*

---

### Description

Calculates the representation of the training classes in depth space.

The detailed descriptions are found in the corresponding topics.

### Usage

```
depth.space.(data, cardinalities, notion, ...)

## Mahalanobis depth
# depth.space.Mahalanobis(data, cardinalities, mah.estimate = "moment", mah.parMcd = 0.75)

## projection depth
# depth.space.projection(data, cardinalities, method = "random", num.directions = 1000)

## Tukey depth
# depth.space.halfspace(data, cardinalities, exact, alg, num.directions = 1000)

## spatial depth
# depth.space.spatial(data, cardinalities)

## zonoid depth
# depth.space.zonoid(data, cardinalities)

# Potential
# depth.space.potential(data, cardinalities, pretransform = "NMom",
#                       kernel = "GKernel", kernel.bandwidth = NULL, mah.parMcd = 0.75)
```

### Arguments

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
notion	The name of the depth notion (shall also work with <a href="#">Custom Methods</a> ).
...	Additional parameters passed to the depth functions.

### Value

Matrix of objects, each object (row) is represented via its depths (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

**See Also**

[depth.space.Mahalanobis](#)  
[depth.space.projection](#)  
[depth.space.halfspace](#)  
[depth.space.spatial](#)  
[depth.space.zonoid](#)

**Examples**

```

# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(10, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(10, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
# Get depth space using zonoid depth
depth.space.(data, c(10, 10), notion = "zonoid")

```

---

depth.space.halfspace *Calculate Depth Space using Halfspace Depth*

---

**Description**

Calculates the representation of the training classes in depth space using the halfspace depth.

**Usage**

```
depth.space.halfspace(data, cardinalities, exact, method, num.directions = 1000, seed = 0)
```

**Arguments**

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
exact	The type of the used method. The default is exact=F, which leads to approximate computation of the halfspace depth. For exact=F, method="Sunif.1D" is used by default. If exact=T, the halfspace depth is computed exactly, with method="recursive" by default.
method	For exact=F, if method="Sunif.1D" (by default), the halfspace depth is computed approximately by being minimized over univariate projections (see details).

For `exact=T`, the halfspace depth is calculated as the minimum over all combinations of  $k$  points from data (see details). In this case parameter `method` specifies  $k$ , with possible values 1 for `method="recursive"` (by default),  $d - 2$  for `method="plane"`,  $d - 1$  for `method="line"`.

The name of the method may be given as well as just parameter `exact`, in which case the default method will be used.

`num.directions` Number of random directions to be generated. As the same direction set is used for all observations, the algorithmic complexity of calculating the depth of each single point in data is logarithmic in the number of observations in data, given the number of directions, see Mozharovskyi et al. (2015), Section 2.3 for discussion.

`seed` The random seed. The default value `seed=0` makes no changes.

### Details

The depth representation is calculated in the same way as in [depth.halfspace](#), see References below for more information and details.

### Value

Matrix of objects, each object (row) is represented via its depths (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

### References

- Cuesta-Albertos, J.A. and Nieto-Reyes, A. (2008). The random Tukey depth. *Computational Statistics and Data Analysis* **52** 4979–4988.
- Dyckerhoff, R. and Mozharovskyi, P. (2016). Exact computation of the halfspace depth. *Computational Statistics and Data Analysis* **98** 19–30.
- Mozharovskyi, P., Mosler, K., and Lange, T. (2015). Classifying real-world data with the  $DD\alpha$ -procedure. *Advances in Data Analysis and Classification* **9** 287–314.
- Rousseeuw, P.J. and Ruts, I. (1996). Algorithm AS 307: Bivariate location depth. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **45** 516–526.
- Tukey, J.W. (1974). Mathematics and the picturing of data. In: *Proceeding of the International Congress of Mathematicians*, Vancouver, 523–531.

### See Also

[ddalpha.train](#) and [ddalpha.classify](#) for application, [depth.halfspace](#) for calculation of the Tukey depth.

### Examples

```
# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(10, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
```

```

class2 <- mvrnorm(10, c(1,1),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
plot(data, col = c(rep(1,10), rep(2,10)))
# Get depth space using the random Tukey depth
dhA = depth.space.halfspace(data, c(10, 10))
(dhA)

# Get depth space using default exact method - "recursive"
dhE = depth.space.halfspace(data, c(10, 10), exact = TRUE)
(dhE)

data <- getdata("hemophilia")
cardinalities = c(sum(data$gr == "normal"), sum(data$gr == "carrier"))
depth.space.halfspace(data[,1:2], cardinalities)

```

---

depth.space.Mahalanobis

*Calculate Depth Space using Mahalanobis Depth*

---

### Description

Calculates the representation of the training classes in depth space using Mahalanobis depth.

### Usage

```
depth.space.Mahalanobis(data, cardinalities, mah.estimate = "moment", mah.parMcd = 0.75)
```

### Arguments

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
mah.estimate	is a character string specifying which estimates to use when calculating the Mahalanobis depth; can be "moment" or "MCD", determining whether traditional moment or Minimum Covariance Determinant (MCD) (see <a href="#">covMcd</a> ) estimates for mean and covariance are used. By default "moment" is used.
mah.parMcd	is the value of the argument alpha for the function <a href="#">covMcd</a> ; is used when mah.estimate = "MCD".

### Details

The depth representation is calculated in the same way as in [depth.Mahalanobis](#), see 'References' for more information and details.

**Value**

Matrix of objects, each object (row) is represented via its depths (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

**References**

- Mahalanobis, P. (1936). On the generalized distance in statistics. *Proceedings of the National Academy India* **12** 49–55.
- Liu, R.Y. (1992). Data depth and multivariate rank tests. In: Dodge, Y. (ed.), *L1-Statistics and Related Methods*, North-Holland (Amsterdam), 279–294.
- Lopuhaa, H.P. and Rousseeuw, P.J. (1991). Breakdown points of affine equivariant estimators of multivariate location and covariance matrices. *The Annals of Statistics* **19** 229–248.
- Rousseeuw, P.J. and Leroy, A.M. (1987). *Robust Regression and Outlier Detection*. John Wiley & Sons (New York).
- Zuo, Y.J. and Serfling, R. (2000). General notions of statistical depth function. *The Annals of Statistics* **28** 461–482.

**See Also**

[ddalpha.train](#) and [ddalpha.classify](#) for application, [depth.Mahalanobis](#) for calculation of Mahalanobis depth.

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(10, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(10, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
# Get depth space using Mahalanobis depth
depth.space.Mahalanobis(data, c(10, 10))
depth.space.Mahalanobis(data, c(10, 10), mah.estimate = "MCD", mah.parMcd = 0.75)

data <- getdata("hemophilia")
cardinalities = c(sum(data$gr == "normal"), sum(data$gr == "carrier"))
depth.space.Mahalanobis(data[,1:2], cardinalities)
```

---

depth.space.potential *Calculate Potential Space*

---

**Description**

Calculates the representation of the training classes in potential space.

**Usage**

```
depth.space.potential(data, cardinalities, pretransform = "NMom",
                      kernel = "GKernel", kernel.bandwidth = NULL, mah.parMcd = 0.75)
```

**Arguments**

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
pretransform	The method of data scaling. NULL to use the original data, The data may be scaled jointly or separately: 1Mom or 1MCD for joint scaling of the classes, NMom or NMCD for separate scaling of the classes. You may use traditional moments or Minimum Covariance Determinant (MCD) estimates for mean and covariance: 1Mom or NMom for scaling using traditional data moments, 1MCD or NMCD for scaling using robust MCD data moments.
kernel	"EDKernel" for the kernel of type $1/(1+kernel.bandwidth*EuclidianDistance2(x, y))$ , "GKernel" [default and recommended] for the simple Gaussian kernel, "EKernel" exponential kernel: $\exp(-kernel.bandwidth*EuclidianDistance(x, y))$ , "VarGKernel" variable Gaussian kernel, where kernel.bandwidth is proportional to the depth.zonoid of a point.
kernel.bandwidth	the bandwidth parameter of the kernel. If NULL - the Scott's rule of thumb is used. May be a single value for all classes, or a vector of values for each of the classes.
mah.parMcd	is the value of the argument alpha for the function <code>covMcd</code> ; is used when pretransform = "*MCD".

**Details**

The potential representation is calculated in the same way as in [depth.potential](#), see References below for more information and details.

**Value**

Matrix of objects, each object (row) is represented via its potentials (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

**References**

Aizerman, M.A., Braverman, E.M., and Rozonoer, L.I. (1970). *The Method of Potential Functions in the Theory of Machine Learning*. Nauka (Moscow).

Pokotylo, O. and Mosler, K. (2015). Classification with the pot-pot plot. *Mimeo*.

**See Also**

[ddalpha.train](#) and [ddalpha.classify](#) for application, [depth.potential](#) for calculation of the potential.

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(50, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(50, c(1,1),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
plot(data, col = c(rep(1,50), rep(2,50)))
# potential with rule of thumb bandwidth
ds = depth.space.potential(data, c(50, 50))
# draw.ddplot(depth.space = ds, cardinalities = c(50, 50))

# potential with bandwidth = 0.5 and joint scaling
ds = depth.space.potential(data, c(50, 50), kernel.bandwidth = 0.5,
                          pretransform = "1Mom")
# draw.ddplot(depth.space = ds, cardinalities = c(50, 50))

# potential with bandwidth = 0.5 and separate scaling
ds = depth.space.potential(data, c(50, 50), kernel.bandwidth = 0.5,
                          pretransform = "NahMom") # or without pretransform
# draw.ddplot(depth.space = ds, cardinalities = c(50, 50))

data <- getdata("hemophilia")
cardinalities = c(sum(data$gr == "normal"), sum(data$gr == "carrier"))
ds = depth.space.potential(data[,1:2], cardinalities)
# draw.ddplot(depth.space = ds, cardinalities = cardinalities)
```

---

depth.space.projection

*Calculate Depth Space using Projection Depth*

---

**Description**

Calculates the representation of the training classes in depth space using projection depth.

**Usage**

```
depth.space.projection(data, cardinalities,
                      method = "random", num.directions = 1000, seed = 0)
```

**Arguments**

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
method	to be used in calculations. "random" Here the depth is determined as the minimum univariate depth of the data projected on lines in several directions. The directions are distributed uniformly on the $(d - 1)$ -sphere; the same direction set is used for all points. "linearize" The Nelder-Mead method for function minimization, taken from Olsson, Journal of Quality Technology, 1974, 6, 56. R-codes of this function were written by Subhajit Dutta.
num.directions	Number of random directions to be generated for method = "random". With the growth of $n$ the complexity grows linearly for the same number of directions.
seed	the random seed. The default value seed=0 makes no changes.

**Details**

The depth representation is calculated in the same way as in [depth.projection](#), see 'References' for more information and details.

**Value**

Matrix of objects, each object (row) is represented via its depths (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

**References**

- Donoho, D.L. (1982). *Breakdown properties of multivariate location estimators*. Ph.D. qualifying paper. Department of Statistics, Harvard University.
- Liu, R.Y. (1992). Data depth and multivariate rank tests. In: Dodge, Y. (ed.), *L1-Statistics and Related Methods*, North-Holland (Amsterdam), 279–294.
- Liu, X. and Zuo, Y. (2014). Computing projection depth and its associated estimators. *Statistics and Computing* **24** 51–63.
- Stahel, W.A. (1981). *Robust estimation: infinitesimal optimality and covariance matrix estimators*. Ph.D. thesis (in German). Eidgenossische Technische Hochschule Zurich.
- Zuo, Y.J. and Lai, S.Y. (2011). Exact computation of bivariate projection depth and the Stahel-Donoho estimator. *Computational Statistics and Data Analysis* **55** 1173–1179.

**See Also**

[ddalpha.train](#) and [ddalpha.classify](#) for application, [depth.projection](#) for calculation of projection depth.

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(10, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(10, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
# Get depth space using projection depth
depth.space.projection(data, c(10, 10), method = "random", num.directions = 1000)
depth.space.projection(data, c(10, 10), method = "linearize")

data <- getdata("hemophilia")
cardinalities = c(sum(data$gr == "normal"), sum(data$gr == "carrier"))
depth.space.projection(data[,1:2], cardinalities)
```

---

depth.space.simplicial

*Calculate Depth Space using Simplicial Depth*

---

**Description**

Calculates the representation of the training classes in depth space using simplicial depth.

**Usage**

```
depth.space.simplicial(data, cardinalities, exact = F, k = 0.05, seed = 0)
```

**Arguments**

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
exact	exact=F (by default) implies the approximative algorithm, considering $k$ simplices, exact=T implies the exact algorithm.
k	Number ( $k > 1$ ) or portion (if $0 < k < 1$ ) of simplices that are considered if exact=F. If $k > 1$ , then the algorithmic complexity is polynomial in $d$ but is independent of the number of observations in data, given $k$ . If $0 < k < 1$ , then the algorithmic complexity is exponential in the number of observations in data, but the calculation precision stays approximately the same.
seed	The random seed. The default value seed=0 makes no changes.

**Details**

The depth representation is calculated in the same way as in [depth.simplicial](#), see 'References' for more information and details.

**Value**

Matrix of objects, each object (row) is represented via its depths (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

**References**

- Chaudhuri, P. (1996). On a geometric notion of quantiles for multivariate data. *Journal of the American Statistical Association* **91** 862–872.
- Liu, R. Y. (1990). On a notion of data depth based on random simplices. *The Annals of Statistics* **18** 405–414.
- Rousseeuw, P.J. and Ruts, I. (1996). Algorithm AS 307: Bivariate location depth. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **45** 516–526.

**See Also**

[ddalpha.train](#) and [ddalpha.classify](#) for application, [depth.simplicial](#) for calculation of simplicial depth.

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(10, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(10, c(1,1),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
# Get depth space using simplicial depth
depth.space.simplicial(data, c(10, 10))

data <- getdata("hemophilia")
cardinalities = c(sum(data$gr == "normal"), sum(data$gr == "carrier"))
depth.space.simplicial(data[,1:2], cardinalities)
```

---

depth.space.simplicialVolume

*Calculate Depth Space using Simplicial Volume Depth*

---

**Description**

Calculates the representation of the training classes in depth space using simplicial volume depth.

**Usage**

```
depth.space.simplicialVolume(data, cardinalities, exact = F, k = 0.05,
                             mah.estimate = "moment", mah.parMcd = 0.75, seed = 0)
```

**Arguments**

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
exact	exact=F (by default) implies the approximative algorithm, considering $k$ simplices, exact=T implies the exact algorithm.
k	Number ( $k > 1$ ) or portion (if $0 < k < 1$ ) of simplices that are considered if exact=F. If $k > 1$ , then the algorithmic complexity is polynomial in $d$ but is independent of the number of observations in data, given $k$ . If $0 < k < 1$ , then the algorithmic complexity is exponential in the number of observations in data, but the calculation precision stays approximately the same.
mah.estimate	A character string specifying affine-invariance adjustment; can be "none", "moment" or "MCD", determining whether no affine-invariance adjustment or moment or Minimum Covariance Determinant (MCD) (see <a href="#">covMcd</a> ) estimates of the covariance are used. By default "moment" is used.
mah.parMcd	The value of the argument alpha for the function <a href="#">covMcd</a> ; is used when mah.estimate = "MCD".
seed	The random seed. The default value seed=0 makes no changes.

**Details**

The depth representation is calculated in the same way as in [depth.simplicialVolume](#), see References below for more information and details.

**Value**

Matrix of objects, each object (row) is represented via its depths (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

**References**

- Oja, H. (1983). Descriptive statistics for multivariate distributions. *Statistics & Probability Letters* **1** 327–332.
- Zuo, Y.J. and Serfling, R. (2000). General notions of statistical depth function. *The Annals of Statistics* **28** 461–482.

**See Also**

[ddalpha.train](#) and [ddalpha.classify](#) for application, [depth.simplicialVolume](#) for calculation of simplicial depth.

## Examples

```
# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(10, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(10, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
# Get depth space using Oja depth
depth.space.simplicialVolume(data, c(10, 10))

data <- getdata("hemophilia")
cardinalities = c(sum(data$gr == "normal"), sum(data$gr == "carrier"))
depth.space.simplicialVolume(data[,1:2], cardinalities)
```

---

depth.space.spatial     *Calculate Depth Space using Spatial Depth*

---

## Description

Calculates the representation of the training classes in depth space using spatial depth.

## Usage

```
depth.space.spatial(data, cardinalities, mah.estimate = "moment", mah.parMcd = 0.75)
```

## Arguments

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
mah.estimate	is a character string specifying which estimates to use when calculating sample covariance matrix; can be "none", "moment" or "MCD", determining whether traditional moment or Minimum Covariance Determinant (MCD) (see <a href="#">covMcd</a> ) estimates for mean and covariance are used. By default "moment" is used. With "none" the non-affine invariant version of Spatial depth is calculated
mah.parMcd	is the value of the argument alpha for the function <a href="#">covMcd</a> ; is used when mah.estimate = "MCD".

## Details

The depth representation is calculated in the same way as in [depth.spatial](#), see 'References' for more information and details.

**Value**

Matrix of objects, each object (row) is represented via its depths (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

**References**

Chaudhuri, P. (1996). On a geometric notion of quantiles for multivariate data. *Journal of the Americal Statistical Association* **91** 862–872.

Koltchinskii, V.I. (1997). M-estimation, convexity and quantiles. *The Annals of Statistics* **25** 435–477.

Serfling, R. (2006). Depth functions in nonparametric multivariate inference. In: Liu, R., Serfling, R., Souvaine, D. (eds.), *Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications*, American Mathematical Society, 1–16.

Vardi, Y. and Zhang, C.H. (2000). The multivariate L1-median and associated data depth. *Proceedings of the National Academy of Sciences, U.S.A.* **97** 1423–1426.

**See Also**

[ddalpha.train](#) and [ddalpha.classify](#) for application, [depth.spatial](#) for calculation of spatial depth.

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(10, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(10, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
# Get depth space using spatial depth
depth.space.spatial(data, c(10, 10))

data <- getdata("hemophilia")
cardinalities = c(sum(data$gr == "normal"), sum(data$gr == "carrier"))
depth.space.spatial(data[,1:2], cardinalities)
```

---

depth.space.zonoid      *Calculate Depth Space using Zonoid Depth*

---

**Description**

Calculates the representation of the training classes in depth space using zonoid depth.

**Usage**

```
depth.space.zonoid(data, cardinalities, seed = 0)
```

**Arguments**

data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
seed	the random seed. The default value seed=0 makes no changes.

**Details**

The depth representation is calculated in the same way as in [depth.zonoid](#), see 'References' for more information and details.

**Value**

Matrix of objects, each object (row) is represented via its depths (columns) w.r.t. each of the classes of the training sample; order of the classes in columns corresponds to the one in the argument cardinalities.

**References**

Dyckerhoff, R., Koshevoy, G., and Mosler, K. (1996). Zonoid data depth: theory and computation. In: Prat A. (ed), *COMPSTAT 1996. Proceedings in computational statistics*, Physica-Verlag (Heidelberg), 235–240.

Koshevoy, G. and Mosler, K. (1997). Zonoid trimming for multivariate distributions *Annals of Statistics* **25** 1998–2017.

Mosler, K. (2002). *Multivariate dispersion, central regions and depth: the lift zonoid approach* Springer (New York).

**See Also**

[ddalpha.train](#) and [ddalpha.classify](#) for application, [depth.zonoid](#) for calculation of zonoid depth.

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 20 training objects
class1 <- mvrnorm(10, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(10, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
# Get depth space using zonoid depth
depth.space.zonoid(data, c(10, 10))

data <- getdata("hemophilia")
cardinalities = c(sum(data$gr == "normal"), sum(data$gr == "carrier"))
depth.space.zonoid(data[,1:2], cardinalities)
```

---

depth.spatial	<i>Calculate Spatial Depth</i>
---------------	--------------------------------

---

**Description**

Calculates the spatial depth of points w.r.t. a multivariate data set.

**Usage**

```
depth.spatial(x, data, mah.estimate = "moment", mah.parMcd = 0.75)
```

**Arguments**

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
mah.estimate	is a character string specifying which estimates to use when calculating sample covariance matrix; can be "none", "moment" or "MCD", determining whether traditional moment or Minimum Covariance Determinant (MCD) (see <a href="#">covMcd</a> ) estimates for mean and covariance are used. By default "moment" is used. With "none" the non-affine invariant version of Spatial depth is calculated
mah.parMcd	is the value of the argument alpha for the function <a href="#">covMcd</a> ; is used when mah.estimate = "MCD".

**Details**

Calculates spatial depth. Spatial depth (also L1-depth) is a distance-based depth exploiting the idea of spatial quantiles of Chaudhuri (1996) and Koltchinskii (1997), formulated by Vardi & Zhang (2000) and Serfling (2002).

**Value**

Numerical vector of depths, one for each row in x; or one depth value if x is a numerical vector.

**References**

- Chaudhuri, P. (1996). On a geometric notion of quantiles for multivariate data. *Journal of the Americal Statistical Association* **91** 862–872.
- Koltchinskii, V.I. (1997). M-estimation, convexity and quantiles. *The Annals of Statistics* **25** 435–477.
- Serfling, R. (2006). Depth functions in nonparametric multivariate inference. In: Liu, R., Serfling, R., Souvaine, D. (eds.), *Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications*, American Mathematical Society, 1–16.

Vardi, Y. and Zhang, C.H. (2000). The multivariate L1-median and associated data depth. *Proceedings of the National Academy of Sciences, U.S.A.* **97** 1423–1426.

### See Also

[depth.halfspace](#) for calculation of the Tukey depth.  
[depth.Mahalanobis](#) for calculation of Mahalanobis depth.  
[depth.projection](#) for calculation of projection depth.  
[depth.simplicial](#) for calculation of simplicial depth.  
[depth.simplicialVolume](#) for calculation of simplicial volume depth.  
[depth.zonoid](#) for calculation of zonoid depth.  
[depth.potential](#) for calculation of data potential.

### Examples

```
# 5-dimensional normal distribution
data <- mvrnorm(1000, rep(0, 5),
              matrix(c(1, 0, 0, 0, 0,
                      0, 2, 0, 0, 0,
                      0, 0, 3, 0, 0,
                      0, 0, 0, 2, 0,
                      0, 0, 0, 0, 1),
                    nrow = 5))
x <- mvrnorm(10, rep(1, 5),
            matrix(c(1, 0, 0, 0, 0,
                    0, 1, 0, 0, 0,
                    0, 0, 1, 0, 0,
                    0, 0, 0, 1, 0,
                    0, 0, 0, 0, 1),
                  nrow = 5))

depths <- depth.spatial(x, data)
cat("Depths: ", depths, "\n")
```

---

depth.zonoid

*Calculate Zonoid Depth*

---

### Description

Calculates the zonoid depth of points w.r.t. a multivariate data set.

### Usage

```
depth.zonoid(x, data, seed = 0)
```

**Arguments**

x	Matrix of objects (numerical vector as one object) whose depth is to be calculated; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix of data where each row contains a $d$ -variate point, w.r.t. which the depth is to be calculated.
seed	the random seed. The default value seed=0 makes no changes.

**Details**

Calculates zonoid depth (Koshevoy and Mosler, 1997; Mosler, 2002) exactly based on the algorithm of Dyckerhoff, Koshevoy and Mosler (1996), implemented in C++ (and provided) by Rainer Dyckerhoff.

**Value**

Numerical vector of depths, one for each row in x; or one depth value if x is a numerical vector.

**References**

Dyckerhoff, R., Koshevoy, G., and Mosler, K. (1996). Zonoid data depth: theory and computation. In: Prat A. (ed), *COMPSTAT 1996. Proceedings in computational statistics*, Physica-Verlag (Heidelberg), 235–240.

Koshevoy, G. and Mosler, K. (1997). Zonoid trimming for multivariate distributions *Annals of Statistics* **25** 1998–2017.

Mosler, K. (2002). *Multivariate dispersion, central regions and depth: the lift zonoid approach* Springer (New York).

**See Also**

[depth.halfspace](#) for calculation of the Tukey depth.

[depth.Mahalanobis](#) for calculation of Mahalanobis depth.

[depth.projection](#) for calculation of projection depth.

[depth.simplicial](#) for calculation of simplicial depth.

[depth.simplicialVolume](#) for calculation of simplicial volume depth.

[depth.spatial](#) for calculation of spatial depth.

[depth.potential](#) for calculation of data potential.

**Examples**

```
# 5-dimensional normal distribution
data <- mvrnorm(1000, rep(0, 5),
               matrix(c(1, 0, 0, 0, 0,
                        0, 2, 0, 0, 0,
                        0, 0, 3, 0, 0,
                        0, 0, 0, 2, 0,
```

```

                                0, 0, 0, 0, 1),
                                nrow = 5))
x <- mvrnorm(10, rep(1, 5),
            matrix(c(1, 0, 0, 0, 0,
                    0, 1, 0, 0, 0,
                    0, 0, 1, 0, 0,
                    0, 0, 0, 1, 0,
                    0, 0, 0, 0, 1),
                  nrow = 5))

depths <- depth.zonoid(x, data)
cat("Depths: ", depths, "\n")

```

---

depthf. *Calculate Functional Depth*

---

### Description

Calculates the depth of functions w.r.t. a functional data set.  
 The detailed descriptions are found in the corresponding topics.

### Usage

```

depthf.(datafA, datafB, notion, ...)

## Adjusted band depth
# depthf.ABD(datafA, datafB, range = NULL, d = 101, norm = c("C", "L2"),
# J = 2, K = 1)

## Band depth
# depthf.BD(datafA, datafB, range = NULL, d = 101)

## Univariate integrated and infimal depth
# depthf.fd1(datafA, datafB, range = NULL, d = 101, order = 1, approx = 0)

## Bivariate integrated and infimal depth
# depthf.fd2(datafA, datafB, range = NULL, d = 101)

## h-mode depth
# depthf.hM(datafA, datafB, range = NULL, d = 101, norm = c("C", "L2"),
# q = 0.2)

## Bivariate h-mode depth
# depthf.hM2(datafA, datafB, range = NULL, d = 101, q = 0.2)

## Half-region depth
# depthf.HR(datafA, datafB, range = NULL, d = 101)

```

```
## Univariate random projection depths
# depthf.RP1(datafA, datafB, range = NULL, d = 101, nproj = 50, nproj2 = 5)

# Bivariate random projection depths
# depthf.RP2(datafA, datafB, range = NULL, d = 101, nproj = 51)
```

### Arguments

datafA	Functions whose depth is computed, represented by a dataf object of their arguments and functional values.
datafB	Random sample functions with respect to which the depth of datafA is computed. datafB is represented by a dataf object of their arguments and functional values.
notion	The name of the depth notion (shall also work with a user-defined depth function named "depthf.<name>").
...	Additional parameters passed to the depth functions.

### Value

Numerical vector of depths, one for each function in datafA; or one depth value if datafA is a single function.

### See Also

[depthf.ABD](#)  
[depthf.BD](#)  
[depthf.fd1](#)  
[depthf.fd2](#)  
[depthf.hM](#)  
[depthf.hM2](#)  
[depthf.HR](#)  
[depthf.RP1](#)  
[depthf.RP2](#)

### Examples

```
# real data example
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]

depthf.(datafA, datafB, notion = "HR")

dataf2A = derivatives.est(datafA,deriv=c(0,1))
dataf2B = derivatives.est(datafB,deriv=c(0,1))

depthf.(dataf2A, dataf2B, notion = "fd2")
```

depthf.ABD

*Adjusted Band Depth for Functional Data***Description**

The adjusted band depth of functional real-valued data based on either the  $C$  (uniform) norm, or on the  $L^2$  norm of functions.

**Usage**

```
depthf.ABD(datafA, datafB, range = NULL, d = 101, norm = c("C", "L2"),
           J = 2, K = 1)
```

**Arguments**

datafA	Functions whose depth is computed, represented by a dataf object of their arguments and functional values. m stands for the number of functions.
datafB	Random sample functions with respect to which the depth of datafA is computed. datafB is represented by a dataf object of their arguments and functional values. n is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length d corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation, see Nagy et al. (2016).
norm	The norm used for the computation of the depth. Two possible choices are implemented: C for the uniform norm of continuous functions, and L2 for the $L^2$ norm of integrable functions.
J	The order of the adjusted band depth, that is the maximal number of functions taken in a band. Acceptable values are 2, 3,... By default this value is set to 2. Note that this is NOT the order as defined in the order-extended version of adjusted band depths in Nagy et al. (2016), used for the detection of shape outlying curves.
K	Number of sub-samples of the functions from B taken to speed up the computation. By default, sub-sampling is not performed. Values of K larger than 1 result in an approximation of the adjusted band depth.

**Details**

The function returns the vector of the sample adjusted band depth values. The kernel used in the evaluation is the function  $K(u) = \exp(-u)$ .

**Value**

A vectors of length  $m$  of the adjusted band depths.

**Author(s)**

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**References**

Gijbels, I., Nagy, S. (2015). Consistency of non-integrated depths for functional data. *Journal of Multivariate Analysis* **140**, 259–282.

Nagy, S., Gijbels, I. and Hlubinka, D. (2016). Weak convergence of discretely observed functional data with applications. *Journal of Multivariate Analysis*, **146**, 46–62.

Nagy, S., Gijbels, I. and Hlubinka, D. (2017). Depth-based recognition of shape outlying functions. *Journal of Computational and Graphical Statistics*, **26** (4), 883–893.

**See Also**

[depthf.BD](#)

**Examples**

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]
depthf.ABD(datafA,datafB)
depthf.ABD(datafA,datafB,norm="L2")
```

---

depthf.BD

*Band Depth for Functional Data*

---

**Description**

The (unadjusted) band depth for functional real-valued data of order  $J=2$ .

**Usage**

```
depthf.BD(datafA, datafB, range = NULL, d = 101)
```

**Arguments**

datafA	Functions whose depth is computed, represented by a dataf object of their arguments and functional values. $m$ stands for the number of functions.
datafB	Random sample functions with respect to which the depth of datafA is computed. datafB is represented by a dataf object of their arguments and functional values. $n$ is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.

range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length d corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.

### Details

The function returns the vector of the sample (unadjusted) band depth values.

### Value

A vector of length m of the band depth values.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### References

Lopez-Pintado, S. and Romo, J. (2009), On the concept of depth for functional data, *J. Amer. Statist. Assoc.* **104** (486), 718 - 734.

### See Also

[depthf.ABD](#), [depthf.f.d1](#)

### Examples

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]
depthf.BD(datafA,datafB)
```

---

depthf.f.d1

*Univariate Integrated and Infimal Depth for Functional Data*

---

### Description

Usual, and order extended integrated and infimal depths for real-valued functional data based on the halfspace and simplicial depth.

### Usage

```
depthf.f.d1(datafA, datafB, range = NULL, d = 101, order = 1, approx = 0)
```

**Arguments**

datafA	Functions whose depth is computed, represented by a dataf object of their arguments and functional values. $m$ stands for the number of functions.
datafB	Random sample functions with respect to which the depth of datafA is computed. datafB is represented by a dataf object of their arguments and functional values. $n$ is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length $d$ corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation, see Nagy et al. (2016).
order	The order of the order extended integrated and infimal depths. By default, this is set to 1, meaning that the usual univariate depths of the functional values are computed. For order=2 or 3, the second and the third order extended integrated and infimal depths are computed, respectively.
approx	Number of approximations used in the computation of the order extended depth for order greater than 1. For order=2, the default value is set to 0, meaning that the depth is computed at all possible $d^{\text{order}}$ combinations of the points in the domain. For order=3, the default value is set to 101. When approx is a positive integer, approx points are randomly sampled in $[0, 1]^{\text{order}}$ and at these points the order-variate depths of the corresponding functional values are computed.

**Details**

The function returns vectors of sample integrated and infimal depth values.

**Value**

Four vectors of length  $m$  of depth values are returned:

- `Simpl_FD` the integrated depth based on the simplicial depth,
- `Half_FD` the integrated depth based on the halfspace depth,
- `Simpl_ID` the infimal depth based on the simplicial depth,
- `Half_ID` the infimal depth based on the halfspace depth.

In addition, two vectors of length  $m$  of the relative area of smallest depth values is returned:

- `Simpl_IA` the proportions of points at which the depth `Simpl_ID` was attained,
- `Half_IA` the proportions of points at which the depth `Half_ID` was attained.

The values `Simpl_IA` and `Half_IA` are always in the interval  $[0,1]$ . They introduce ranking also among functions having the same infimal depth value - if two functions have the same infimal depth, the one with larger infimal area `IA` is said to be less central. For `order=2` and `m=1`, two additional matrices of pointwise depths are also returned:

- `PSD` the matrix of size  $d \times d$  containing the computed pointwise bivariate simplicial depths used for the computation of `Simpl_FD` and `Simpl_ID`,
- `PHD` the matrix of size  $d \times d$  containing the computed pointwise bivariate halfspace depths used for the computation of `Half_FD` and `Half_ID`.

For `order=3`, only `Half_FD` and `Half_ID` are provided.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### References

Nagy, S., Gijbels, I. and Hlubinka, D. (2016). Weak convergence of discretely observed functional data with applications. *Journal of Multivariate Analysis*, **146**, 46–62.

Nagy, S., Gijbels, I. and Hlubinka, D. (2017). Depth-based recognition of shape outlying functions. *Journal of Computational and Graphical Statistics*, **26** (4), 883–893.

### See Also

[depthf.fd2](#), [infimalRank](#)

### Examples

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]
depthf.fd1(datafA,datafB)
depthf.fd1(datafA,datafB,order=2)
depthf.fd1(datafA,datafB,order=3,approx=51)
```

---

depthf.fd2

*Bivariate Integrated and Infimal Depth for Functional Data*

---

### Description

Integrated and infimal depths of functional bivariate data (that is, data of the form  $X : [a, b] \rightarrow R^2$ , or  $X : [a, b] \rightarrow R$  and the derivative of  $X$ ) based on the bivariate halfspace and simplicial depths.

### Usage

```
depthf.fd2(datafA, datafB, range = NULL, d = 101)
```

**Arguments**

datafA	Bivariate functions whose depth is computed, represented by a multivariate dataf object of their arguments (vector), and a matrix with two columns of the corresponding bivariate functional values. $m$ stands for the number of functions.
datafB	Bivariate random sample functions with respect to which the depth of datafA is computed. datafB is represented by a multivariate dataf object of their arguments (vector), and a matrix with two columns of the corresponding bivariate functional values. $n$ is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length $d$ corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.

**Details**

The function returns the vectors of sample integrated and infimal depth values.

**Value**

Four vectors of length  $m$  are returned:

- `Simpl_FD` the integrated depth based on the bivariate simplicial depth,
- `Half_FD` the integrated depth based on the bivariate halfspace depth,
- `Simpl_ID` the infimal depth based on the bivariate simplicial depth,
- `Half_ID` the infimal depth based on the bivariate halfspace depth.

In addition, two vectors of length  $m$  of the relative area of smallest depth values is returned:

- `Simpl_IA` the proportions of points at which the depth `Simpl_ID` was attained,
- `Half_IA` the proportions of points at which the depth `Half_ID` was attained.

The values `Simpl_IA` and `Half_IA` are always in the interval  $[0,1]$ . They introduce ranking also among functions having the same infimal depth value - if two functions have the same infimal depth, the one with larger infimal area IA is said to be less central.

**Author(s)**

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**References**

- Hlubinka, D., Gijbels, I., Omelka, M. and Nagy, S. (2015). Integrated data depth for smooth functions and its application in supervised classification. *Computational Statistics*, **30** (4), 1011–1031.
- Nagy, S., Gijbels, I. and Hlubinka, D. (2017). Depth-based recognition of shape outlying functions. *Journal of Computational and Graphical Statistics*, **26** (4), 883–893.

**See Also**

[depthf.fd1](#), [infimalRank](#)

**Examples**

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]

dataf2A = derivatives.est(datafA,deriv=c(0,1))
dataf2B = derivatives.est(datafB,deriv=c(0,1))
depthf.fd2(dataf2A,dataf2B)
```

---

depthf.hM

*h-Mode Depth for Functional Data*

---

**Description**

The h-mode depth of functional real-valued data.

**Usage**

```
depthf.hM(datafA, datafB, range = NULL, d = 101, norm = c("C", "L2"),
  q = 0.2)
```

**Arguments**

datafA	Functions whose depth is computed, represented by a dataf object of their arguments and functional values. m stands for the number of functions.
datafB	Random sample functions with respect to which the depth of datafA is computed. datafB is represented by a dataf object of their arguments and functional values. n is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length d corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.
norm	The norm used for the computation of the depth. Two possible choices are implemented: C for the uniform norm of continuous functions, and L <sup>2</sup> for the L <sup>2</sup> norm of integrable functions.
q	The quantile used to determine the value of the bandwidth <i>h</i> in the computation of the h-mode depth. <i>h</i> is taken as the q-quantile of all non-zero distances between the functions B. By default, this value is set to q=0.2, in accordance with the choice of Cuevas et al. (2007).

**Details**

The function returns the vectors of the sample h-mode depth values. The kernel used in the evaluation is the standard Gaussian kernel, the bandwidth value is chosen as a quantile of the non-zero distances between the random sample curves.

**Value**

A vector of length  $m$  of the h-mode depth values.

**Author(s)**

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**References**

Cuevas, A., Febrero, M. and Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics* **22** (3), 481–496.

Nagy, S., Gijbels, I. and Hlubinka, D. (2016). Weak convergence of discretely observed functional data with applications. *Journal of Multivariate Analysis*, **146**, 46–62.

**Examples**

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]
depthf.hM(datafA,datafB)
depthf.hM(datafA,datafB,norm="L2")
```

---

depthf.hM2

*Bivariate h-Mode Depth for Functional Data Based on the  $L^2$  Metric*

---

**Description**

The h-mode depth of functional bivariate data (that is, data of the form  $X : [a, b] \rightarrow R^2$ , or  $X : [a, b] \rightarrow R$  and the derivative of  $X$ ) based on the  $L^2$  metric of functions.

**Usage**

```
depthf.hM2(datafA, datafB, range = NULL, d = 101, q = 0.2)
```

**Arguments**

`datafA` Bivariate functions whose depth is computed, represented by a multivariate `dataf` object of their arguments (vector), and a matrix with two columns of the corresponding bivariate functional values. `m` stands for the number of functions.

datafB	Bivariate random sample functions with respect to which the depth of datafA is computed. datafB is represented by a multivariate dataf object of their arguments (vector), and a matrix with two columns of the corresponding bivariate functional values. n is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length d corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.
q	The quantile used to determine the value of the bandwidth $h$ in the computation of the h-mode depth. $h$ is taken as the q-quantile of all non-zero distances between the functions B. By default, this value is set to $q=0.2$ , in accordance with the choice of Cuevas et al. (2007).

### Details

The function returns the vectors of sample h-mode depth values. The kernel used in the evaluation is the standard Gaussian kernel, the bandwidth value is chosen as a quantile of the non-zero distances between the random sample curves.

### Value

Three vectors of length m of h-mode depth values are returned:

- hM the unscaled h-mode depth,
- hM\_norm the h-mode depth hM linearly transformed so that its range is [0,1],
- hM\_norm2 the h-mode depth FD linearly transformed by a transformation such that the range of the h-mode depth of B with respect to B is [0,1]. This depth may give negative values.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### References

Cuevas, A., Febrero, M. and Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics* **22** (3), 481–496.

### See Also

[depthf.hM](#)

**Examples**

```

datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]

datafA2 = derivatives.est(datafA,deriv=c(0,1))
datafB2 = derivatives.est(datafB,deriv=c(0,1))

depthf.hM2(datafA2,datafB2)

depthf.hM2(datafA2,datafB2)$hM
# depthf.hM2(cbind(A2[,1],A2[,2]),cbind(B2[,1],B2[,2]))$hM
# the two expressions above should give the same result

```

---

depthf.HR

*Half-Region Depth for Functional Data*


---

**Description**

The half-region depth for functional real-valued data.

**Usage**

```
depthf.HR(datafA, datafB, range = NULL, d = 101)
```

**Arguments**

datafA	Functions whose depth is computed, represented by a dataf object of their arguments and functional values. <i>m</i> stands for the number of functions.
datafB	Random sample functions with respect to which the depth of datafA is computed. datafB is represented by a dataf object of their arguments and functional values. <i>n</i> is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length <i>d</i> corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.

**Details**

The function returns the vector of the sample half-region depth values.

**Value**

A vector of length  $m$  of the half-region depth values.

**Author(s)**

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**References**

Lopez-Pintado, S. and Romo, J. (2011). A half-region depth for functional data. *Computational Statistics & Data Analysis* **55** (4), 1679–1695.

**Examples**

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]
depthf.HR(datafA,datafB)
```

---

depthf.RP1

*Univariate Random Projection Depths for Functional Data*

---

**Description**

Random projection depth and random functional depth for functional data.

**Usage**

```
depthf.RP1(datafA, datafB, range = NULL, d = 101, nproj = 50, nproj2 = 5)
```

**Arguments**

datafA	Functions whose depth is computed, represented by a dataf object of their arguments and functional values. $m$ stands for the number of functions.
datafB	Random sample functions with respect to which the depth of datafA is computed. datafB is represented by a dataf object of their arguments and functional values. $n$ is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length $d$ corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.
nproj	Number of projections taken in the computation of the random projection depth. By default taken to be 51.

`nproj2` Number of projections taken in the computation of the random functional depth. By default taken to be 5. `nproj2` should be much smaller than `d`, the dimensionality of the discretized functional data.

### Details

The function returns the vectors of sample random projection, and random functional depth values. The random projection depth described in Cuevas et al. (2007) is based on the average univariate depth of one-dimensional projections of functional data. The projections are taken randomly as a sample of standard normal  $d$ -dimensional random variables, where  $d$  stands for the dimensionality of the discretized functional data.

The random functional depth (also called random Tukey depth, or random halfspace depth) is described in Cuesta-Albertos and Nieto-Reyes (2008). The functional data are projected into the real line in random directions as for the random projection depths. Afterwards, an approximation of the halfspace (Tukey) depth based on this limited number of univariate projections is assessed.

### Value

Three vectors of depth values of length  $m$  are returned:

- `Simpl_FD` the random projection depth based on the univariate simplicial depth,
- `Half_FD` the random projection depth based on the univariate halfspace depth,
- `RHalf_FD` the random halfspace depth.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### References

Cuevas, A., Febrero, M. and Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions, *Computational Statistics* **22** (3), 481–496.

Cuesta-Albertos, J.A. and Nieto-Reyes, A. (2008). The random Tukey depth. *Computational Statistics & Data Analysis* **52** (11), 4979–4988.

### See Also

[depthf.RP2](#)

### Examples

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]

depthf.RP1(datafA,datafB)
```

depthf.RP2

*Bivariate Random Projection Depths for Functional Data***Description**

Double random projection depths of functional bivariate data (that is, data of the form  $X : [a, b] \rightarrow R^2$ , or  $X : [a, b] \rightarrow R$  and the derivative of  $X$ ).

**Usage**

```
depthf.RP2(datafA, datafB, range = NULL, d = 101, nproj = 51)
```

**Arguments**

datafA	Bivariate functions whose depth is computed, represented by a multivariate dataf object of their arguments (vector), and a matrix with two columns of the corresponding bivariate functional values. $m$ stands for the number of functions.
datafB	Bivariate random sample functions with respect to which the depth of datafA is computed. datafB is represented by a multivariate dataf object of their arguments (vector), and a matrix with two columns of the corresponding bivariate functional values. $n$ is the sample size. The grid of observation points for the functions datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length $d$ corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.
nproj	Number of projections taken in the computation of the double random projection depth. By default taken to be 51.

**Details**

The function returns the vectors of sample double random projection depth values. The double random projection depths are described in Cuevas et al. (2007). They are of two types: RP2 type, and RPD type. Both types of depths are based on bivariate projections of the bivariate functional data. These projections are taken randomly as a sample of standard normal  $d$ -dimensional random variables, where  $d$  stands for the dimensionality of the internally represented discretized functional data. For RP2 type depths, the average bivariate depth of the projected quantities is assessed. For RPD type depths, further univariate projections of these bivariate projected quantities are evaluated, and based on these final univariate quantities, the average univariate depth is computed.

**Value**

Five vectors of length  $m$  are returned:

- `Simpl_FD` the double random projection depth RP2 based on the bivariate simplicial depth,
- `Half_FD` the double random projection depth RP2 based on the bivariate halfspace depth,
- `hM_FD` the double random projection depth RP2 based on the bivariate h-mode depth,
- `Simpl_DD` the double random projection depth RPD based on the univariate simplicial depth,
- `Half_DD` the random projection depth RPD based on the univariate halfspace depth,

**Author(s)**

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**References**

Cuevas, A., Febrero, M. and Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics* **22** (3), 481–496.

**See Also**

[depthf.RP1](#)

**Examples**

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]

dataf2A = derivatives.est(datafA,deriv=c(0,1))
dataf2B = derivatives.est(datafB,deriv=c(0,1))
depthf.RP2(dataf2A,dataf2B)
```

---

`depthf.simplicialBand` *Calculate Simplicial Band Depth*

---

**Description**

Calculate the simplicial band depth defined by Lopez-Pintado, Sun, Lin, Genton (2014).

**Usage**

```
depthf.simplicialBand(objectsf, dataf, modified = TRUE, J = NULL,
                      range = NULL, d = 101)
```

**Arguments**

objectsf	Functoins for which the depth should be computed; a list containing lists (functions) of two vectors of equal length, named args and vals: arguments sorted in ascending order and corresponding them values respectively.
dataf	Data sample of functoins w.r.t. which the depth should be computed; structure as for objectsf.
modified	Whether modified simplicial band depth should be computed; logical, TRUE by default.
J	How many functions to consider in each tuple of the U-statistics; integer, d+1 by default.
range	The common range of the domain where the functions of objectsf and dataf are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in objectsf and dataf.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length d corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.

**Value**

A vector of depths of each of objectsf w.r.t. dataf.

**References**

Lopez-Pintado, Sun, Lin, Genton (2014). Simplicial band depth for multivariate data. *Advances in Data Analysis and Classification* **8**(3) 321–338.

**See Also**

[depthf.BD](#), [depthf.ABD](#).

**Examples**

```
datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]

dataf2A = derivatives.est(datafA,deriv=c(0,1))
dataf2B = derivatives.est(datafB,deriv=c(0,1))
depthf.simplicialBand(dataf2A,dataf2B)
```

derivatives.est

*Estimation of the First Two Derivatives for Functional Data***Description**

Returns the estimated values of derivatives of functional data.

**Usage**

```
derivatives.est(dataf, range = NULL, d = 101, spar = NULL, deriv = c(0,1))
```

**Arguments**

dataf	Functional dataset, represented by a dataf object of their arguments and functional values. $m$ stands for the number of functions.
range	The common range of the domain where the functions dataf are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in dataf.
d	Grid size to which all the functional data are transformed. For computation, all functional observations are first transformed into vectors of their functional values of length $d$ corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.
spar	If provided, this parameter is passed to functions D1ss and D2ss from package sfsmisc as the value of the smoothing spline parameter in order to numerically approximate the derivatives of dataf.
deriv	A vector composed of 0, 1, and 2 of the demanded functional values / derivatives of the functions in the rows of dataf. 0 stands for the functional values, 1 for the first derivatives, 2 for the second derivatives.

**Details**

If the input dataf is a functional random sample of size  $m$ , the function returns a dataf object of  $nd$ -dimensional functional data, where in the elements of the vector-valued functional data represent the estimated values of the derivatives of dataf. All derivatives are evaluated at an equi-distant grid of  $d$  points in the domain given by range.  $nd$  here stands for 1, 2 or 3, depending on how many derivatives of dataf are requested to be computed. For the estimation, functions D1ss and D2ss from the package sfsmisc are utilized.

**Value**

A multivariate dataf object of the functional values and / or the derivatives of dataf. The dimensionality of the vector-valued functional data is  $nd$ . The arguments of the data are all equal to an equi-distant grid of  $d$  points in the domain given by range.  $nd$  is the demanded number of derivatives at the output, i.e. the length of the vector deriv.

**Author(s)**

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**See Also**

[D1ss](#) in package **sfsmisc**

[D2ss](#) in package **sfsmisc**

**Examples**

```
dataf = dataf.population()$dataf
derivatives.est(dataf,deriv=c(0,1,2))
```

---

dknn.classify

*Depth-Based kNN*

---

**Description**

The implementation of the affine-invariant depth-based kNN of Paidaveine and Van Bever (2015).

**Usage**

```
dknn.classify(objects, data, k, depth = "halfspace", seed = 0)
```

**Arguments**

objects	Matrix containing objects to be classified; each row is one $d$ -dimensional object.
data	Matrix containing training sample where each of $n$ rows is one object of the training sample where first $d$ entries are inputs and the last entry is output (class label).
k	the number of neighbours
depth	Character string determining which depth notion to use; the default value is "halfspace". Currently the method supports the following depths: "halfspace", "Mahalanobis", "simplicial".
seed	the random seed. The default value seed=0 makes no changes.

**Value**

List containing class labels, or character string "Ignored" for the outsiders if "Ignore" was specified as the outsider treating method.

**References**

Paindaveine, D. and Van Bever, G. (2015). Nonparametrically consistent depth-based classifiers. *Bernoulli* **21** 62–82.

**See Also**

[dknn.train](#) to train the Dknn-classifier.

[dknn.classify.trained](#) to classify with the Dknn-classifier.

[ddalpha.train](#) to train the  $DD\alpha$ -classifier.

[ddalpha.getErrorRateCV](#) and [ddalpha.getErrorRatePart](#) to get error rate of the Dknn-classifier on particular data (set separator = "Dknn").

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 200 training objects and 200 to test with
class1 <- mvrnorm(200, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
trainIndices <- c(1:100)
testIndices <- c(101:200)
propertyVars <- c(1:2)
classVar <- 3
trainData <- rbind(cbind(class1[trainIndices,], rep(1, 100)),
                  cbind(class2[trainIndices,], rep(2, 100)))
testData <- rbind(cbind(class1[testIndices,], rep(1, 100)),
                  cbind(class2[testIndices,], rep(2, 100)))
data <- list(train = trainData, test = testData)

# Train the classifier
# and get the classification error rate
cls <- dknn.train(data$train, kMax = 20, depth = "Mahalanobis")
cls$k
classes1 <- dknn.classify.trained(data$test[,propertyVars], cls)
cat("Classification error rate: ",
    sum(unlist(classes1) != data$test[,classVar])/200)

# Classify the new data based on the old ones in one step
classes2 <- dknn.classify(data$test[,propertyVars], data$train, k = cls$k, depth = "Mahalanobis")
cat("Classification error rate: ",
    sum(unlist(classes2) != data$test[,classVar])/200)
```

---

dknn.classify.trained *Depth-Based kNN*

---

**Description**

The implementation of the affine-invariant depth-based kNN of Paidaveine and Van Bever (2015).

**Usage**

```
dknn.classify.trained(objects, dknn)
```

**Arguments**

`objects` Matrix containing objects to be classified; each row is one  $d$ -dimensional object.  
`dknn` Dknn-classifier (obtained by [dknn.train](#)).

**Value**

List containing class labels, or character string "Ignored" for the outsiders if "Ignore" was specified as the outsider treating method.

**References**

Paindaveine, D. and Van Bever, G. (2015). Nonparametrically consistent depth-based classifiers. *Bernoulli* **21** 62–82.

**See Also**

[dknn.train](#) to train the Dknn-classifier.

[dknn.classify](#) to classify with the Dknn-classifier.

[ddalpha.train](#) to train the  $DD\alpha$ -classifier.

[ddalpha.getErrorRateCV](#) and [ddalpha.getErrorRatePart](#) to get error rate of the Dknn-classifier on particular data (set separator = "Dknn").

**Examples**

```
# Generate a bivariate normal location-shift classification task
# containing 200 training objects and 200 to test with
class1 <- mvrnorm(200, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
trainIndices <- c(1:100)
testIndices <- c(101:200)
propertyVars <- c(1:2)
classVar <- 3
trainData <- rbind(cbind(class1[trainIndices,], rep(1, 100)),
                  cbind(class2[trainIndices,], rep(2, 100)))
testData <- rbind(cbind(class1[testIndices,], rep(1, 100)),
                  cbind(class2[testIndices,], rep(2, 100)))
data <- list(train = trainData, test = testData)

# Train the classifier
# and get the classification error rate
cls <- dknn.train(data$train, kMax = 20, depth = "Mahalanobis")
cls$k
classes1 <- dknn.classify.trained(data$test[,propertyVars], cls)
```

```

cat("Classification error rate: ",
    sum(unlist(classes1) != data$test[,classVar])/200)

# Classify the new data based on the old ones in one step
classes2 <- dknn.classify(data$test[,propertyVars], data$train, k = cls$k, depth = "Mahalanobis")
cat("Classification error rate: ",
    sum(unlist(classes2) != data$test[,classVar])/200)

```

---

dknn.train

*Depth-Based kNN*


---

### Description

The implementation of the affine-invariant depth-based kNN of Paindaveine and Van Bever (2015).

### Usage

```
dknn.train(data, kMax = -1, depth = "halfspace", seed = 0)
```

### Arguments

data	Matrix containing training sample where each of $n$ rows is one object of the training sample where first $d$ entries are inputs and the last entry is output (class label).
kMax	the maximal value for the number of neighbours. If the value is set to -1, the default value is calculated as $n/2$ , but at least 2, at most $n-1$ .
depth	Character string determining which depth notion to use; the default value is "halfspace". Currently the method supports the following depths: "halfspace", "Mahalanobis", "simplicial".
seed	the random seed. The default value <code>seed=0</code> makes no changes.

### Value

The returned object contains technical information for classification, including the found optimal value  $k$ .

### References

Paindaveine, D. and Van Bever, G. (2015). Nonparametrically consistent depth-based classifiers. *Bernoulli* **21** 62–82.

### See Also

[dknn.classify](#) and [dknn.classify.trained](#) to classify with the Dknn-classifier.

[ddalpha.train](#) to train the  $DD\alpha$ -classifier.

[ddalpha.getErrorRateCV](#) and [ddalpha.getErrorRatePart](#) to get error rate of the Dknn-classifier on particular data (set separator = "Dknn").

## Examples

```
# Generate a bivariate normal location-shift classification task
# containing 200 training objects and 200 to test with
class1 <- mvrnorm(200, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
trainIndices <- c(1:100)
testIndices <- c(101:200)
propertyVars <- c(1:2)
classVar <- 3
trainData <- rbind(cbind(class1[trainIndices,], rep(1, 100)),
                  cbind(class2[trainIndices,], rep(2, 100)))
testData <- rbind(cbind(class1[testIndices,], rep(1, 100)),
                  cbind(class2[testIndices,], rep(2, 100)))
data <- list(train = trainData, test = testData)

# Train the classifier
# and get the classification error rate
cls <- dknn.train(data$train, kMax = 20, depth = "Mahalanobis")
cls$k
classes1 <- dknn.classify.trained(data$test[,propertyVars], cls)
cat("Classification error rate: ",
    sum(unlist(classes1) != data$test[,classVar])/200)

# Classify the new data based on the old ones in one step
classes2 <- dknn.classify(data$test[,propertyVars], data$train, k = cls$k, depth = "Mahalanobis")
cat("Classification error rate: ",
    sum(unlist(classes2) != data$test[,classVar])/200)
```

---

draw.ddplot

*Draw DD-Plot*

---

## Description

The function draws the *DD*-plot either of the existing  $DD_\alpha$ -classifier of the depth space. Also accessible from [plot.ddalpha](#).

## Usage

```
draw.ddplot(ddalpha, depth.space, cardinalities,
            main = "DD plot", xlab = "C1", ylab = "C2", xlim, ylim,
            classes = c(1, 2), colors = c("red", "blue", "green"), drawsep = T)
```

**Arguments**

<code>ddalpha</code>	DD $\alpha$ -classifier (obtained by <code>ddalpha.train</code> ).
<code>depth.space</code>	The ready depth space obtained by <code>depth.space</code> .
<code>cardinalities</code>	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
<code>main</code>	an overall title for the plot: see <code>title</code>
<code>xlab, ylab</code>	class labels
<code>xlim, ylim</code>	range of axis
<code>classes</code>	vector of numbers of two classes used for depth calculation
<code>colors</code>	vector of the classes' colors
<code>drawsep</code>	draws the separation on the DD-plot (currently for 2 classes and not for knn)

**See Also**

[ddalpha.train](#)  
[depth.space](#).

**Examples**

```
data = getdata("kidney")

#1. using the existing ddalpha classifier
ddalpha = ddalpha.train(data, depth = "spatial")
draw.ddplot(ddalpha, main = "DD-plot")

#2. using depth.space.
# Sort the data w.r.t. classes
data = rbind(data[data$C == 1,], data[data$C == 2,])
cardinalities = c(sum(data$C == 1), sum(data$C == 2))

dspace = depth.space.spatial(data[, -6], cardinalities = cardinalities)
draw.ddplot(depth.space = dspace, cardinalities = cardinalities,
            main = "DD-plot", xlab = 1, ylab = 2)
```

**Description**

Produces a kernel smoothed version of a function based on the vectors given in the input. Bandwidth is selected using cross-validation.

**Usage**

```
FKS(dataf, Tout, kernel = c("uniform", "triangular", "Epanechnikov",
    "biweight", "triweight", "Gaussian"), m = 51, K = 20)
```

**Arguments**

<code>dataf</code>	A set of functional data given by a <code>dataf</code> object that are to be smoothed.
<code>Tout</code>	vector of values in the domain of the functions at which the resulting smoothed function is evaluated
<code>kernel</code>	Kernel used for smoothing. Admissible values are <code>uniform</code> , <code>triangular</code> , <code>Epanechnikov</code> , <code>biweight</code> , <code>triweight</code> and <code>Gaussian</code> . By default, <code>uniform</code> is used.
<code>m</code>	Number of points in the grid for choosing the cross-validated bandwidth.
<code>K</code>	Performs K-fold cross-validation based on randomly shuffled data.

**Details**

A vector of the same length as `Tout` corresponding to the values of the function produced using kernel smoothing, is provided. Bandwidth is selected using the K-fold cross-validation of randomly shuffled input values.

**Value**

A `dataf` object corresponding to `Tout` of smoothed functional values.

**Author(s)**

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**Examples**

```
d = 10
T = sort(runif(d))
X = T^2 + rnorm(d, sd=.1)
Tout = seq(0, 1, length=101)

plot(T, X)
dataf = list(list(args=T, vals=X))
data.sm = FKS(dataf, Tout, kernel="Epan")
lines(data.sm[[1]]$args, data.sm[[1]]$vals, col=2)

datafs = structure(list(dataf=dataf, labels=1:length(dataf)), class="functional")
plot(datafs)
points(T, X)
data.sms = structure(list(dataf=data.sm, labels=1:length(data.sm)), class="functional")
plot(data.sms)

n = 6
dataf = list()
for(i in 1:n) dataf[[i]] = list(args = T<-sort(runif(d)), vals = T^2 + rnorm(d, sd=.1))
data.sm = FKS(dataf, Tout, kernel="triweight")
data.sms = structure(list(dataf=data.sm, labels=1:length(data.sm)), class="functional")
plot(data.sms)
```

---

`getdata`*Data for Classification*

---

### Description

50 multivariate data sets for binary classification. For more details refer <https://wisostat.uni-koeln.de/de/forschung/software-und-daten/data-for-classification/>

The `getdata` function gets the data set from the package, and returns it. The dataset itself does not appear in the global environment and the existing variables with the same name remain unchanged.

### Usage

```
# load the data set
# data(name)

# load the data set by name
# data(list = "name")

# load the data set by name to a variable
# getdata("name")
```

### Arguments

`name`                    the data set name.

### Format

A data frame with  $n$  observations on the  $d$  variables. The last  $d+1$  column is the class label.

`x[, 1:d]` numeric values

`x[, d+1]` the numeric class label (0 or 1) or (1 or 2)

### Details

The package contains data sets used in the joint project of the University of Cologne and the Hochschule Merseburg "Classifying real-world data with the DD $\alpha$ -procedure". Comprehensive description of the methodology, and experimental settings and results of the study are presented in the work:

Mozharovskiy, P., Mosler, K., and Lange, T. (2015). Classifying real-world data with the DD $\alpha$ -procedure. *Advances in Data Analysis and Classification* **9** 287–314.

For a more complete explanation of the technique and further experiments see: Lange, T., Mosler, K., and Mozharovskiy, P. (2014). Fast nonparametric classification based on data depth. *Statistical Papers* **55** 49–69.

50 binary classification tasks have been obtained from partitioning 33 freely accessible data sets. Multiclass problems were reasonably split into binary classification problems, some of the data set

were slightly processed by removing objects or attributes and selecting prevailing classes. Each data set is provided with a (short) description and brief descriptive statistics. The name reflects the origination of the data. A letter after the name is a property filter, letters (also their combinations) in brackets separated by "vs" are the classes opposed. The letters (combinations or words) stand for labels of classes (names of properties) and are intuitive. Each description contains a link to the original data.

The data have been collected as open source data in January 2013. Owners of the package decline any responsibility regarding their correctness or consequences of their usage. If you publish material based on these data, please quote the original source. Special requests regarding citations are found on data set's web page.

## Note

List of the datasets:

baby  
banknoten  
biomed  
bloodtransfusion  
breast\_cancer\_wisconsin  
bupa  
chemdiab\_1vs2  
chemdiab\_1vs3  
chemdiab\_2vs3  
cloud  
crabB\_MvsF  
crabF\_BvsO  
crabM\_BvsO  
crabO\_MvsF  
crab\_BvsO  
crab\_MvsF  
cricket\_CvsP  
diabetes  
ecoli\_cpvsim  
ecoli\_cpvspp  
ecoli\_imvspp  
gemsen\_MvsF  
glass  
groessen\_MvsF  
haberman  
heart  
hemophilia  
indian\_liver\_patient\_1vs2  
indian\_liver\_patient\_FvsM  
iris\_setosavsversicolor  
iris\_setosavsvirginica  
iris\_versicolorsvirginica  
irish\_ed\_MvsF  
kidney

pima  
plasma\_retinol\_MvsF  
segmentation  
socmob\_IvsNI  
socmob\_WvsB  
tae  
tennis\_MvsF  
tips\_DvsN  
tips\_MvsF  
uscrime\_SvsN  
vertebral\_column  
veteran\_lung\_cancer  
vowel\_MvsF  
wine\_1vs2  
wine\_1vs3  
wine\_2vs3

Also functional data sets can be loaded:

geneexp  
growth  
medflies  
population  
population2010  
tecator

## References

Lange, T., Mosler, K., and Mozharovskyi, P. (2014). Fast nonparametric classification based on data depth. *Statistical Papers* **55** 49–69.

Mozharovskyi, P., Mosler, K., and Lange, T. (2015). Classifying real-world data with the  $DD\alpha$ -procedure. *Advances in Data Analysis and Classification* **9** 287–314.

The general list of sources consists of:

UCI Machine Learning Repository <https://archive.ics.uci.edu/ml/>  
R-packages <https://CRAN.R-project.org/>  
<https://www.stat.cmu.edu/>  
<https://stat.ethz.ch/Teaching/Datasets/>  
<https://www.stats.ox.ac.uk/pub/PRNN/>

## See Also

[utils:data](#)

## Examples

```
# load a dataset using data()
data(hemophilia)
data(list = "hemophilia")
rm(hemophilia)
```

```
# load data set using getdata()
hemophilia = "This is some existing object called 'hemophilia'. It remains unchanged"
d = getdata("hemophilia")
head(d)
print(hemophilia)

#get the list of all data sets
names = data(package = "ddalpha")$results[,3]
```

---

infimalRank

*Adjusted Ranking of Functional Data Based on the Infimal Depth*


---

### Description

Returns a vector of adjusted depth-based ranks for infimal depth for functional data.

### Usage

```
infimalRank(ID, IA, ties.method = "max")
```

### Arguments

ID	The vector of infimal depths of the curves of length n.
IA	The vector of the infimal areas corresponding to the infimal depths from ID of length n.
ties.method	Parameter for breaking ties in infimal area index. By default max, see rank.

### Details

Infimal depths for functional data tend to give to many functional observations the same value of depth. Using this function, the data whose depth is the same is ranked according to the infimal area indicator. This indicator is provided in functions `depthf.f.d1` along the value of the infimal depth.

### Value

A vector of length n. Low depth values mean high ranks, i.e. potential outlyingness. If some of the infimal depths are identical, the ranking of these functions is made according to the values of the infimal area. There, higher infimal area index means higher rank, i.e. non-centrality.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### References

Nagy, S., Gijbels, I. and Hlubinka, D. (2017). Depth-based recognition of shape outlying functions. *Journal of Computational and Graphical Statistics*, **26** (4), 883–893.

**Examples**

```

datafA = dataf.population()$dataf[1:20]
datafB = dataf.population()$dataf[21:50]
D = depthf.fd1(datafA,datafB)
infimalRank(D$Half_ID,D$Half_IA)

ID = c(0,1,0,0,0,1,1)
IA = c(2,3,1,0,2,4,1)
infimalRank(ID,IA)

```

---

is.in.convex

*Check Outsiderness*


---

**Description**

Checks the belonging to at least one of class convex hulls of the training sample.

**Usage**

```
is.in.convex(x, data, cardinalities, seed = 0)
```

**Arguments**

x	Matrix of objects (numerical vector as one object) whose belonging to convex hulls is to be checked; each row contains a $d$ -variate point. Should have the same dimension as data.
data	Matrix containing training sample where each row is a $d$ -dimensional object, and objects of each class are kept together so that the matrix can be thought of as containing blocks of objects, representing classes.
cardinalities	Numerical vector of cardinalities of each class in data, each entry corresponds to one class.
seed	the random seed. The default value seed=0 makes no changes.

**Details**

Checks are conducted w.r.t. each separate class in data using the simplex algorithm, taken from the C++ implementation of the zonoid depth calculation by Rainer Dyckerhoff.

**Value**

Matrix of number of objects rows and number of classes columns, containing 1 if an object belongs to the convex hull of the corresponding class, and 0 otherwise.

**Author(s)**

Implementation of the simplex algorithm is taken from the algorithm for computation of zonoid depth (Dyckerhoff, Koshevoy and Mosler, 1996) that has been implemented in C++ by Rainer Dyckerhoff.

## References

Dyckerhoff, R., Koshevoy, G., and Mosler, K. (1996). Zonoid data depth: theory and computation. In: Prat A. (ed), *COMPSTAT 1996. Proceedings in computational statistics*, Physica-Verlag (Heidelberg), 235–240.

## See Also

[ddalpha.train](#) and [ddalpha.classify](#) for application.

## Examples

```
# Generate a bivariate normal location-shift classification task
# containing 400 training objects and 1000 to test with
class1 <- mvrnorm(700, c(0,0),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(700, c(2,2),
                 matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
trainIndices <- c(1:200)
testIndices <- c(201:700)
propertyVars <- c(1:2)
classVar <- 3
trainData <- rbind(cbind(class1[trainIndices,], rep(1, 200)),
                  cbind(class2[trainIndices,], rep(2, 200)))
testData <- rbind(cbind(class1[testIndices,], rep(1, 500)),
                  cbind(class2[testIndices,], rep(2, 500)))
data <- list(train = trainData, test = testData)

# Count outsiders
numOutsiders = sum(rowSums(is.in.convex(data$test[,propertyVars],
                                       data$train[,propertyVars], c(200, 200))) == 0)
cat(numOutsiders, "outsiders found in the testing sample.\n")
```

---

L2metric

*Fast Computation of the  $L^2$  Metric for Sets of Functional Data*

---

## Description

Returns the matrix of  $L^2$  distances between two sets of functional data.

## Usage

```
L2metric(A, B)
```

## Arguments

A Functions of the first set, represented by a matrix of their functional values of size  $m \times d$ .  $m$  stands for the number of functions,  $d$  is the number of the equi-distant points  $1, \dots, d$  in the domain of the data  $[1, d]$  at which the functional values of the  $m$  functions are evaluated.

B Functions of the second set, represented by a matrix of their functional values of size  $n \times d$ .  $n$  stands for the number of functions,  $d$  is the number of the equidistant points  $1, \dots, d$  in the domain of the data  $[1, d]$  at which the functional values of the  $n$  functions are evaluated. The grid of observation points for the functions A and B must be the same.

### Details

For two sets of functional data of sizes  $m$  and  $n$  represented by matrices of their functional values on the common domain  $1, \dots, d$ , this function returns the symmetric matrix of size  $m \times n$  whose entry in the  $i$ -th row and  $j$ -th column is the approximated  $L^2$  distance of the  $i$ -th function from the first set, and the  $j$ -th function from the second set. This function is utilized in the computation of the  $h$ -mode depth.

### Value

A symmetric matrix of the distances of the functions of size  $m \times n$ .

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### See Also

[depthf.hm](#)  
[dataf2rawfd](#)

### Examples

```
datapop = dataf2rawfd(dataf.population())$dataf, range=c(1950, 2015), d=66)
A = datapop[1:20, ]
B = datapop[21:50, ]
L2metric(A, B)
```

---

plot.ddalpha

*Plots for the "ddalpha" Class*

---

### Description

[depth.contours.ddalpha](#) – builds the data depth contours for multiclass 2-dimensional data using the trained classifier. [draw.ddplot](#) – draws the  $DD$ -plot of the existing  $DD_\alpha$ -classifier.

### Usage

```
## S3 method for class 'ddalpha'
plot(x, type = c("ddplot", "depth.contours"), ...)
```

**Arguments**

`x`  $DD\alpha$ -classifier (obtained by `ddalpha.train`).  
`type` type of the plot for `draw.ddplot` or `depth.contours.ddalpha`  
`...` additional parameters passed to the depth functions and to `plot`

**See Also**

[depth.](#)  
[depth.contours](#)  
[depth.graph](#)

**Examples**

```
## Not run:

par(mfrow = c(2,2))
data(hemophilia)

ddalpha = ddalpha.train(hemophilia, depth = "none")
plot(ddalpha, type = "depth.contours", main = "data")
plot(ddalpha, type = "ddplot", main = "data", drawsep = F)

for (depth in c("zonoid", "Mahalanobis", "projection", "spatial")){
  ddalpha = ddalpha.train(hemophilia, depth = depth)
  plot(ddalpha, type = "depth.contours", main = depth, drawsep = T)
  plot(ddalpha, type = "ddplot", main = depth)
}

## End(Not run)
```

---

plot.ddalphaf

*Plots for the "ddalphaf" Class*


---

**Description**

[plot.functional](#) – plots the functional data used by classifier  
[depth.contours.ddalpha](#) – builds the data depth contours for multiclass 2-dimensional data using the trained classifier. [draw.ddplot](#) – draws the  $DD$ -plot of the existing  $DD\alpha$ -classifier.

**Usage**

```
## S3 method for class 'ddalphaf'
plot(x, type = c("functional.data", "ddplot", "depth.contours"), ...)
```

**Arguments**

x functional  $DD\alpha$ -classifier (obtained by `ddalphaf.train`).  
 type type of the plot for `plot.functional`, `draw.ddplot` or `depth.contours.ddalpha`  
 ... additional parameters passed to the depth functions and to `plot`

**See Also**

[depth.](#)  
[depth.contours](#)  
[depth.graph](#)

**Examples**

```
## Not run:

dataf = dataf.growth()
ddalphaf = ddalphaf.train (dataf$dataf, dataf$labels,
                          classifier.type = "ddalpha", maxNumIntervals = 2)

# plot the functional data
plot(ddalphaf)

# plot depth contours and separation in the transformed space
# (possible only if maxNumIntervals = 2)
plot(ddalphaf, type = "depth.contours")

# plot the DD-plot
plot(ddalphaf, type = "ddplot")

## End(Not run)
```

---

plot.functional      *Plot functions for the Functional Data*

---

**Description**

Plots the functional data given in the form which is described in the topic `dataf.*`.

**Usage**

```
## S3 method for class 'functional'
plot(x,
     main = "Functional data", xlab = "args", ylab = "vals",
     colors = c("red", "blue", "green", "black", "orange", "pink"), ...)
```

```
## S3 method for class 'functional'
lines(x,
      colors = c("red", "blue", "green", "black", "orange", "pink"), ...)

## S3 method for class 'functional'
points(x,
       colors = c("red", "blue", "green", "black", "orange", "pink"), ...)
```

### Arguments

x	The functional data as in the topic <a href="#">dataf.*</a> . Note, that the in order to use s3 methods the data must be of class "functional".
main	an overall title for the plot: see <a href="#">title</a>
xlab	a title for the x axis: see <a href="#">title</a>
ylab	a title for the y axis: see <a href="#">title</a>
colors	the colors for the classes of the data. The colors are applied to the classes sorted in alphabetical order. Use the same set of classes to ensure that the same colours are selected in lines and points as in plot (do not remove entire classes).
...	additional parameters

### See Also

[dataf.\\*](#) for functional data description

### Examples

```
## Not run:
## load the Growth dataset
dataf = dataf.growth()

labels = unlist(dataf$labels)
plot(dataf,
     main = paste("Growth: girls red (", sum(labels == "girl"), ")",
                  " boys blue (", sum(labels == "boy"), ")", sep=""),
     xlab="Year", ylab="Height, cm",
     colors = c("blue", "red") # in alphabetical order of class labels
)

# plot an observation as a line
observation = structure(list(dataf = list(dataf$dataf[[1]])), class = "functional")
lines(observation, colors = "green", lwd = 3)

# plot hight at the age of 14
indexAge14 = which(observation$dataf[[1]]$args == 14)
hightAge14 = observation$dataf[[1]]$vals[indexAge14]
atAge14 = structure(list(
  dataf = list(dataf = list(args = 14, vals = hightAge14))
), class = "functional")
points(atAge14, colors = "yellow", pch = 18)
```

```
## End(Not run)
```

---

`rawfd2dataf`*Transform Raw Functional Data to a dataf Object*

---

### Description

Constructs a (possibly multivariate) functional data object given by an array of its functional values evaluated at an equi-distant grid of points, and transforms it into a `dataf` object more suitable for work in the `ddalpha` package.

### Usage

```
rawfd2dataf(X, range)
```

### Arguments

<code>X</code>	Either a matrix of size $n \times d$ , or an array of dimension $n \times d \times k$ of functional values. Here $n$ stands for the number of functions, $d$ is the number of equi-distant points in the domain where the functional values are evaluated, and if applicable, $k$ is the dimensionality of the (vector-valued) functional data.
<code>range</code>	A vector of size two that represents the endpoints of the common domain of all functions $X$ .

### Value

A (possibly multivariate) `dataf` object corresponding to the functional data  $X$  evaluated at an equi-distant grid of points.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### See Also

[dataf2rawfd](#)

[depthf.fd1](#)

[depthf.fd2](#)

## Examples

```
## transform a matrix into a functional data set
n = 5
d = 21
X = matrix(rnorm(n*d), ncol=d)
rawfd2dataf(X, range=c(0, 1))

## transform an array into a multivariate functional data set
k = 3
X = array(rnorm(n*d*k), dim=c(n,d,k))
rawfd2dataf(X, range=c(-1, 1))
```

---

resetPar

*Reset Graphical Parameters*

---

## Description

The function returns the default graphical parameters for [par](#).

## Usage

```
resetPar()
```

## Details

The returned parameters are used as input parameters for [par](#).

## Value

The list of graphical parameters described in the 'Graphical Parameters' section of [par](#).

## Examples

```
par(mfrow = c(1,2), mar = c(2,2,2,2))
plot(sin, -pi, 2*pi)
plot(cos, -pi, 2*pi)

par(resetPar())
plot(sin, -pi, 2*pi)
plot(cos, -pi, 2*pi)
```

---

shape.fd.analysis	<i>Diagnostic Plot for First and Second Order Integrated and Infimal Depths</i>
-------------------	---

---

### Description

Produce the diagnostic plot based on the first or second order extended integrated / infimal depths.

### Usage

```
shape.fd.analysis(datafA, datafB, range = NULL, d = 101, order = 1,
  method = c("halfspace", "simplicial"), approx = 0, title = "",
  nfun = 10, plot = TRUE)
```

### Arguments

datafA	A single function whose depth is computed, represented by a dataf object of arguments and functional values.
datafB	Functional dataset with respect to which the depth of datafA is computed. datafB is represented by a dataf object of arguments and functional values. n stands for the number of functions. The grid of observation points for the functions in datafA and datafB may not be the same.
range	The common range of the domain where the functions datafA and datafB are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in datafA and datafB.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length d corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.
order	The order of the depth to be used in the plot, for order=1 produces the plot of univariate marginal depth of A and nfun functions from B over the domain of the functions. For order=2 produces the bivariate contour plot of the bivariate depths of A at couples of points from the domain.
method	The depth that is used in the diagnostic plot. possible values are halfspace for the halfspace depth, or simplicial for the simplicial depth.
approx	For order=2, the number of approximations used in the computation of the order extended depth. By default this is set to 0, meaning that the depth is computed at all possible $d^2$ combinations of the points in the domain. When set to a positive integer, approx bivariate points are randomly sampled in unit square, and at these points the bivariate depths of the corresponding functional values are computed.
title	The title of the diagnostic plot.

nfun	For order=1, the number of functions from B whose coordinate-wise univariate depths of functional values should be displayed with the depth of A. The depth of A is displayed in solid red line, the depths of the functions from B in dashed black.
plot	Logical: should the function by plotted?

### Details

Plots a diagnostic plot of pointwise univariate (or bivariate) depths for all possible points (or couples of points) from the domain of the functional data. From such a plot it is possible to infer into the first order (or second order) properties of a single function  $x$  with respect to the given set of functional data. For order=1, the integral of the displayed function is the integrated depth of  $x$ , the smallest value of the function is the infimal depth of  $x$ . For order=2, the bivariate integral of the displayed surface gives the second order extended integrated depth of  $x$ , the infimum of this bivariate function gives the second order infimal depth of  $x$ . For details see Nagy et al. (2016) and [depthf.fd1](#).

### Value

For order=1 two depth values, and two vectors of pointwise depths:

- `Simpl_FD` the first order integrated depth based on the simplicial depth,
- `Half_FD` the first order integrated depth based on the halfspace depth,
- `Simpl_ID` the first order infimal depth based on the simplicial depth,
- `Half_ID` the first order infimal depth based on the halfspace depth,
- `PSD` the vector of length `d` containing the computed pointwise univariate simplicial depths used for the computation of `Simpl_FD` and `Simpl_ID`,
- `PHD` the vector of length `d` containing the computed pointwise univariate halfspace depths used for the computation of `Half_FD` and `Half_ID`.

In addition, the first order integrated / infimal depth diagnostic plot of the function A with respect to the random sample given by the functions corresponding to the rows of the matrix B is produced.

For order=2 four depth values, and two matrices of pointwise depths:

- `Simpl_FD` the second order integrated depth based on the simplicial depth,
- `Half_FD` the second order integrated depth based on the halfspace depth,
- `Simpl_ID` the second order infimal depth based on the simplicial depth,
- `Half_ID` the second order infimal depth based on the halfspace depth,
- `PSD` the matrix of size `d*d` containing the computed pointwise bivariate simplicial depths used for the computation of `Simpl_FD` and `Simpl_ID`,
- `PHD` the matrix of size `d*d` containing the computed pointwise bivariate halfspace depths used for the computation of `Half_FD` and `Half_ID`.

In addition, the second order integrated / infimal depth diagnostic plot of the function A with respect to the random sample given by the functions corresponding to the rows of the matrix B is produced.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

**References**

Nagy, S., Gijbels, I. and Hlubinka, D. (2017). Depth-based recognition of shape outlying functions. *Journal of Computational and Graphical Statistics*, **26** (4), 883–893.

**See Also**

[depthf.fd1](#)

**Examples**

```
datafA = dataf.population()$dataf[1]
dataf = dataf.population()$dataf[2:20]
shape.fd.analysis(datafA,dataf,order=1)
shape.fd.analysis(datafA,dataf,order=2,approx=0)
```

---

shape.fd.outliers      *Functional Depth-Based Shape Outlier Detection*

---

**Description**

Detects functional outliers of first three orders, based on the order extended integrated depth for functional data.

**Usage**

```
shape.fd.outliers(dataf, range = NULL, d = 101, q = 0.05,
  method = c("halfspace", "simplicial"), approx = 100, print = FALSE,
  plotpairs = FALSE, max.order = 3, exclude.out = TRUE,
  output = c("matrix", "list"), identifiers = NULL)
```

**Arguments**

dataf	Functional dataset, represented by a dataf object of their arguments and functional values. n stands for the number of functions.
range	The common range of the domain where the functions dataf are observed. Vector of length 2 with the left and the right end of the interval. Must contain all arguments given in dataf.
d	Grid size to which all the functional data are transformed. For depth computation, all functional observations are first transformed into vectors of their functional values of length d corresponding to equi-spaced points in the domain given by the interval range. Functional values in these points are reconstructed using linear interpolation, and extrapolation.
q	The quantile presenting a threshold for the first order outlier detection. Functions with first order integrated depth smaller than the q quantile of this sample of depths are flagged as potential outliers. If set to NULL, the outliers are detected from the first order integrated depth after the log-transformation, as for higher order outliers.

method	The depth that is used in the diagnostic plot. possible values are <code>halfspace</code> for the halfspace depth, or <code>simplicial</code> for the simplicial depth.
approx	For the computation of the third order integrated depth, the number of approximations used in the computation of the order extended depth. By default this is set to 100, meaning that 100 trivariate points are randomly sampled in unit cube, and at these points the trivariate depths of the corresponding functional values. May be set to 0 to compute the depth at all possible $d^3$ combinations of the points in the domain. This choice may result in very slow computation, see also <a href="#">depthf.fd1</a> .
print	If the rows of X are named, <code>print=TRUE</code> enables a graphical output when the names of the outlying curves are displayed.
plotpairs	If set to TRUE, the scatter plot of the computed depths for orders 1, 2 and 3 is displayed. Here, the depths corresponding to the flagged outliers are plotted in colour.
max.order	Maximal order of shape outlyingness to be computed, can be set to 1, 2, or 3.
exclude.out	Logical variable; exclude the detected lower order outliers in the flagging process? By default TRUE.
output	Output method, can be set to <code>matrix</code> for a matrix with logical entries (TRUE for outliers), or <code>list</code> for a list of outliers.
identifiers	A vector of names for the data observations. Facilitates identification of outlying functions.

### Details

Using the procedure described in Nagy et al. (2016), the function uses the order extended integrated depths for functions, see [depthf.fd1](#) and [shape.fd.analysis](#), to perform informal functional shape outlier detection. Outliers of the first order (horizontal shift outliers) are found as the functions with  $q\%$  of smallest (first order) integrated depth values. Second and third order outliers (shape outliers) are found using the extension of the boxplot method for depths as described in the paper Nagy et al. (2016).

### Value

A matrix of logical values of size  $n \times 4$ , where  $n$  is the sample size. In the first three rows indicators of outlyingness of the corresponding functions for orders 1, 2 and 3 are given, in the fourth row the indicator of outlyingness with respect to the comparison of the first, and third order depths is given. That is, the first row corresponds to the first order outliers, the second row to the second order outliers, and the last two rows formally to the third order outliers. Please consult Nagy et al. (2016) to interpret the notion of shape outlyingness.

### Author(s)

Stanislav Nagy, <nagy@karlin.mff.cuni.cz>

### References

Nagy, S., Gijbels, I. and Hlubinka, D. (2017). Depth-based recognition of shape outlying functions. *Journal of Computational and Graphical Statistics*, **26** (4), 883–893.

**See Also**

[depthf.fd1](#), [shape.fd.analysis](#)

**Examples**

```
n = 30
dataf = dataf.population()$dataf[1:n]
shape.fd.outliers(dataf, print=TRUE, plotpairs=TRUE,
  identifiers=unlist(dataf.population()$identifier)[1:n])
```

# Index

- \* **benchmark**
  - ddalpha.getErrorRateCV, 33
  - ddalpha.getErrorRatePart, 34
  - ddalpha.test, 35
  - ddalphaf.getErrorRateCV, 45
  - ddalphaf.getErrorRatePart, 46
  - ddalphaf.test, 47
- \* **classif**
  - compclassf.classify, 7
  - compclassf.train, 8
  - Custom Methods, 10
  - ddalpha-package, 3
  - ddalpha.classify, 31
  - ddalpha.train, 37
  - ddalphaf.classify, 43
  - ddalphaf.train, 49
  - dknn.classify, 110
  - dknn.classify.trained, 111
  - dknn.train, 113
- \* **custom**
  - Custom Methods, 10
- \* **datasets**
  - dataf.\*, 17
  - dataf.geneexp, 18
  - dataf.growth, 19
  - dataf.medflies, 20
  - dataf.population, 22
  - dataf.population2010, 23
  - dataf.sim.1.CFF07, 25
  - dataf.sim.2.CFF07, 26
  - dataf.tecator, 28
  - getdata, 117
- \* **depth**
  - Custom Methods, 10
  - depth.sample, 70
  - depthf.ABD, 94
  - depthf.BD, 95
  - depthf.fd1, 96
  - depthf.fd2, 98
  - depthf.hM, 100
  - depthf.hM2, 101
  - depthf.HR, 103
  - depthf.RP1, 104
  - depthf.RP2, 106
  - infimalRank, 120
  - shape.fd.analysis, 129
  - shape.fd.outliers, 131
- \* **derivatives**
  - depthf.fd2, 98
  - depthf.hM2, 101
  - depthf.RP2, 106
  - derivatives.est, 109
- \* **fdata**
  - dataf, 15
- \* **functional**
  - Cmetric, 6
  - compclassf.classify, 7
  - compclassf.train, 8
  - dataf, 15
  - dataf.\*, 17
  - dataf.geneexp, 18
  - dataf.growth, 19
  - dataf.medflies, 20
  - dataf.population, 22
  - dataf.population2010, 23
  - dataf.sim.1.CFF07, 25
  - dataf.sim.2.CFF07, 26
  - dataf.tecator, 28
  - dataf2rawfd, 29
  - ddalpha-package, 3
  - ddalphaf.classify, 43
  - ddalphaf.train, 49
  - depthf., 92
  - depthf.ABD, 94
  - depthf.BD, 95
  - depthf.fd1, 96
  - depthf.fd2, 98
  - depthf.hM, 100

- depthf.hM2, 101
- depthf.HR, 103
- depthf.RP1, 104
- depthf.RP2, 106
- depthf.simplicialBand, 107
- derivatives.est, 109
- FKS, 115
- infimalRank, 120
- L2metric, 122
- plot.ddalphaf, 124
- plot.functional, 125
- rawfd2dataf, 127
- shape.fd.analysis, 129
- shape.fd.outliers, 131
- \* **kernel**
  - derivatives.est, 109
  - FKS, 115
- \* **metric**
  - Cmetric, 6
  - L2metric, 122
- \* **multivariate**
  - compclassf.classify, 7
  - compclassf.train, 8
  - ddalpha-package, 3
  - ddalpha.classify, 31
  - ddalpha.train, 37
  - ddalphaf.classify, 43
  - ddalphaf.train, 49
  - depth., 51
  - depth.betaSkeleton, 53
  - depth.halfspace, 59
  - depth.L2, 61
  - depth.Mahalanobis, 63
  - depth.potential, 65
  - depth.projection, 67
  - depth.qhpeeling, 69
  - depth.simplicial, 71
  - depth.simplicialVolume, 73
  - depth.space., 75
  - depth.space.halfspace, 76
  - depth.space.Mahalanobis, 78
  - depth.space.potential, 79
  - depth.space.projection, 81
  - depth.space.simplicial, 83
  - depth.space.simplicialVolume, 84
  - depth.space.spatial, 86
  - depth.space.zonoid, 87
  - depth.spatial, 89
- depth.zonoid, 90
- dknn.classify, 110
- dknn.classify.trained, 111
- dknn.train, 113
- is.in.convex, 121
- \* **nonparametric**
  - compclassf.classify, 7
  - compclassf.train, 8
  - ddalpha-package, 3
  - ddalpha.classify, 31
  - ddalpha.train, 37
  - ddalphaf.classify, 43
  - ddalphaf.train, 49
  - depth., 51
  - depth.betaSkeleton, 53
  - depth.halfspace, 59
  - depth.L2, 61
  - depth.Mahalanobis, 63
  - depth.potential, 65
  - depth.projection, 67
  - depth.qhpeeling, 69
  - depth.simplicial, 71
  - depth.simplicialVolume, 73
  - depth.space., 75
  - depth.space.halfspace, 76
  - depth.space.Mahalanobis, 78
  - depth.space.potential, 79
  - depth.space.projection, 81
  - depth.space.simplicial, 83
  - depth.space.simplicialVolume, 84
  - depth.space.spatial, 86
  - depth.space.zonoid, 87
  - depth.spatial, 89
  - depth.zonoid, 90
  - depthf., 92
  - depthf.simplicialBand, 107
  - dknn.classify, 110
  - dknn.classify.trained, 111
  - dknn.train, 113
- \* **outlier**
  - shape.fd.analysis, 129
  - shape.fd.outliers, 131
- \* **package**
  - ddalpha-package, 3
- \* **plot**
  - shape.fd.analysis, 129
- \* **rank**
  - infimalRank, 120

**\* robust**

compclassf.classify, 7  
 compclassf.train, 8  
 ddalpha-package, 3  
 ddalpha.classify, 31  
 ddalpha.train, 37  
 ddalphaf.classify, 43  
 ddalphaf.train, 49  
 depth., 51  
 depth.betaSkeleton, 53  
 depth.halfspace, 59  
 depth.L2, 61  
 depth.Mahalanobis, 63  
 depth.potential, 65  
 depth.projection, 67  
 depth.qhpeeling, 69  
 depth.simplicial, 71  
 depth.simplicialVolume, 73  
 depth.space., 75  
 depth.space.halfspace, 76  
 depth.space.Mahalanobis, 78  
 depth.space.potential, 79  
 depth.space.projection, 81  
 depth.space.simplicial, 83  
 depth.space.simplicialVolume, 84  
 depth.space.spatial, 86  
 depth.space.zonoid, 87  
 depth.spatial, 89  
 depth.zonoid, 90  
 depthf., 92  
 depthf.simplicialBand, 107

**\* shape**

shape.fd.analysis, 129

**\* smoothing**

FKS, 115

**\* visualization**

depth.contours, 55  
 depth.contours.ddalpha, 56  
 depth.graph, 58  
 draw.ddplot, 114  
 plot.ddalpha, 123  
 plot.ddalphaf, 124  
 plot.functional, 125  
 resetPar, 128

alpha(ddalpha.train), 37

baby(getdata), 117

banknoten(getdata), 117

biomed(getdata), 117

bloodtransfusion(getdata), 117

breast\_cancer\_wisconsin(getdata), 117

bupa(getdata), 117

chemdiab\_1vs2(getdata), 117

chemdiab\_1vs3(getdata), 117

chemdiab\_2vs3(getdata), 117

cloud(getdata), 117

Cmetric, 6

compclassf.classify, 4, 5, 7, 9

compclassf.train, 4, 5, 7, 8, 8, 16, 45, 46, 48, 50

covMcd, 38, 41, 54, 61–63, 65, 73, 78, 80, 85, 86, 89

crab\_Bvs0(getdata), 117

crab\_MvsF(getdata), 117

crabB\_MvsF(getdata), 117

crabF\_Bvs0(getdata), 117

crabM\_Bvs0(getdata), 117

crab0\_MvsF(getdata), 117

cricket\_CvsP(getdata), 117

Custom Methods, 4, 10, 38, 39, 75

D1ss, 110

D2ss, 110

data(getdata), 117

dataf, 15

dataf.\*, 4, 5, 9, 16, 17, 19–21, 23, 24, 26–28, 50, 125, 126

dataf.geneexp, 17, 18

dataf.growth, 17, 19

dataf.medflies, 17, 20

dataf.population, 17, 22, 24

dataf.population2010, 17, 23, 23

dataf.sim.1.CFF07, 17, 25

dataf.sim.2.CFF07, 17, 26

dataf.tecator, 17, 28

dataf2rawfd, 7, 29, 123, 127

ddalpha(ddalpha-package), 3

ddalpha-package, 3

ddalpha.classify, 4, 5, 31, 33, 35, 36, 42, 77, 79, 81, 83–85, 87, 88, 122

ddalpha.getErrorRateCV, 33, 35, 36, 111–113

ddalpha.getErrorRatePart, 33, 34, 36, 111–113

ddalpha.test, 33, 35, 35

- ddalpha.train, [4](#), [5](#), [10](#), [11](#), [13](#), [31–36](#), [37](#), [57](#), [77](#), [79](#), [81](#), [83–85](#), [87](#), [88](#), [111–113](#), [115](#), [122](#), [124](#)
- ddalphaf.classify, [5](#), [43](#), [45](#), [47](#), [48](#), [50](#)
- ddalphaf.getErrorRateCV, [45](#), [47](#), [48](#)
- ddalphaf.getErrorRatePart, [45](#), [46](#), [48](#)
- ddalphaf.test, [45](#), [47](#), [47](#)
- ddalphaf.train, [4](#), [5](#), [9](#), [15](#), [16](#), [44–48](#), [49](#), [125](#)
- depth., [4](#), [5](#), [42](#), [51](#), [55–58](#), [124](#), [125](#)
- depth.betaSkeleton, [52](#), [53](#)
- depth.contours, [4](#), [55](#), [57](#), [124](#), [125](#)
- depth.contours.ddalpha, [4](#), [56](#), [56](#), [123–125](#)
- depth.graph, [4](#), [5](#), [53](#), [56](#), [57](#), [58](#), [124](#), [125](#)
- depth.halfspace, [39](#), [52](#), [54](#), [59](#), [62](#), [64](#), [66](#), [68](#), [69](#), [71](#), [72](#), [74](#), [77](#), [90](#), [91](#)
- depth.L2, [52](#), [61](#), [69](#)
- depth.Mahalanobis, [39](#), [52](#), [54](#), [60](#), [62](#), [63](#), [66](#), [68](#), [69](#), [72](#), [74](#), [78](#), [79](#), [90](#), [91](#)
- depth.potential, [53](#), [55](#), [60](#), [62](#), [64](#), [65](#), [68](#), [70](#), [72](#), [74](#), [80](#), [81](#), [90](#), [91](#)
- depth.projection, [39](#), [52](#), [54](#), [60](#), [62](#), [64](#), [66](#), [67](#), [69](#), [72](#), [74](#), [82](#), [83](#), [90](#), [91](#)
- depth.qhpeeling, [53](#), [62](#), [69](#)
- depth.sample, [70](#)
- depth.simplicial, [39](#), [53](#), [54](#), [60](#), [62](#), [64](#), [66](#), [68](#), [69](#), [71](#), [71](#), [74](#), [84](#), [90](#), [91](#)
- depth.simplicialVolume, [39](#), [53](#), [54](#), [60](#), [62](#), [64](#), [66](#), [68](#), [69](#), [72](#), [73](#), [85](#), [90](#), [91](#)
- depth.space., [4](#), [5](#), [42](#), [75](#), [115](#)
- depth.space.halfspace, [76](#), [76](#)
- depth.space.Mahalanobis, [76](#), [78](#)
- depth.space.potential, [79](#)
- depth.space.projection, [76](#), [81](#)
- depth.space.simplicial, [83](#)
- depth.space.simplicialVolume, [84](#)
- depth.space.spatial, [76](#), [86](#)
- depth.space.zonoid, [76](#), [87](#)
- depth.spatial, [39](#), [53](#), [54](#), [60](#), [62](#), [64](#), [66](#), [68](#), [70](#), [72](#), [74](#), [86](#), [87](#), [89](#), [91](#)
- depth.zonoid, [39](#), [53](#), [55](#), [60](#), [62](#), [64](#), [66](#), [68](#), [70](#), [72](#), [74](#), [88](#), [90](#), [90](#)
- depthf., [4](#), [5](#), [92](#)
- depthf.ABD, [93](#), [94](#), [96](#), [108](#)
- depthf.BD, [93](#), [95](#), [95](#), [108](#)
- depthf.fd1, [30](#), [93](#), [96](#), [96](#), [100](#), [127](#), [130–133](#)
- depthf.fd2, [30](#), [93](#), [98](#), [98](#), [127](#)
- depthf.hM, [7](#), [93](#), [100](#), [102](#), [123](#)
- depthf.hM2, [93](#), [101](#)
- depthf.HR, [93](#), [103](#)
- depthf.RP1, [93](#), [104](#), [107](#)
- depthf.RP2, [93](#), [105](#), [106](#)
- depthf.simplicialBand, [107](#)
- derivatives.est, [109](#)
- diabetes (getdata), [117](#)
- dknn.classify, [110](#), [112](#), [113](#)
- dknn.classify.trained, [111](#), [111](#), [113](#)
- dknn.train, [111](#), [112](#), [113](#)
- draw.ddplot, [4](#), [5](#), [114](#), [123–125](#)
- ecoli\_cpvsim (getdata), [117](#)
- ecoli\_cpvspp (getdata), [117](#)
- ecoli\_imvspp (getdata), [117](#)
- fdata, [15](#)
- FKS, [115](#)
- gemsen\_MvsF (getdata), [117](#)
- geneexp (dataf.geneexp), [18](#)
- getdata, [4](#), [5](#), [117](#)
- glass (getdata), [117](#)
- groessen\_MvsF (getdata), [117](#)
- growth (dataf.growth), [19](#)
- haberman (getdata), [117](#)
- heart (getdata), [117](#)
- hemophilia (getdata), [117](#)
- indian\_liver\_patient\_1vs2 (getdata), [117](#)
- indian\_liver\_patient\_FvsM (getdata), [117](#)
- infimalRank, [98](#), [100](#), [120](#)
- iris\_setosavsversicolor (getdata), [117](#)
- iris\_setosavsvirginica (getdata), [117](#)
- iris\_versicolorsvirginica (getdata), [117](#)
- irish\_ed\_MvsF (getdata), [117](#)
- is.in.convex, [4](#), [5](#), [42](#), [121](#)
- kidney (getdata), [117](#)
- knn, [40](#)
- knn.cv, [40](#)
- knnlm (ddalpha.train), [37](#)
- L2metric, [122](#)
- lda, [40](#)
- lines.functional (plot.functional), [125](#)

maxD (ddalpha.train), 37  
medflies (dataf.medflies), 20  
outsiders (ddalpha.train), 37  
par, 128  
persp, 58  
pima (getdata), 117  
plasma\_retinol\_MvsF (getdata), 117  
plot, 56, 57, 124, 125  
plot.ddalpha, 5, 56, 114, 123  
plot.ddalphaf, 5, 124  
plot.functional, 4, 5, 16, 17, 19–21, 23, 24,  
26–28, 124, 125, 125  
plot.window, 58  
points.functional (plot.functional), 125  
polynomial (ddalpha.train), 37  
population (dataf.population), 22  
population2010 (dataf.population2010),  
23  
predict.compclassf  
    (compclassf.classify), 7  
predict.ddalpha (ddalpha.classify), 31  
predict.ddalphaf (ddalphaf.classify), 43  
rawfd2dataf, 30, 127  
resetPar, 128  
segmentation (getdata), 117  
shape.fd.analysis, 129, 132, 133  
shape.fd.outliers, 131  
socmob\_IvsNI (getdata), 117  
socmob\_WvsB (getdata), 117  
tae (getdata), 117  
tecator (dataf.tecator), 28  
tennis\_MvsF (getdata), 117  
tips\_DvsN (getdata), 117  
tips\_MvsF (getdata), 117  
title, 55, 57, 58, 115, 126  
uscrime\_SvsN (getdata), 117  
utils:data, 119  
vertebral\_column (getdata), 117  
veteran\_lung\_cancer (getdata), 117  
vowel\_MvsF (getdata), 117  
wine\_1vs2 (getdata), 117  
wine\_1vs3 (getdata), 117  
wine\_2vs3 (getdata), 117