Package ‘ddalpha’
March 28, 2014

Type Package
Title DDalpha-classifier, depth-calculating methods and associated functions
Version 1.0.6
Date 2014-03-14
Depends MASS, class, robustbase
LinkingTo BH
Description Contains methods for depth classification based on the \{\alpha\}-procedure and for calculation of statistical depths
License GPL-2
Author Oleksii Pokotylo [aut, cre], Pavlo Mozharovskyi [aut]
Maintainer Oleksii Pokotylo <alexey.pokotylo@gmail.com>
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-03-28 07:18:41

R topics documented:

  ddalpha-package ............................................. 2
  ddalpha.classify ........................................... 4
  ddalpha.train .............................................. 5
  depth.randomTukey ......................................... 10
  depth.space.randomTukey ................................ 12
  depth.space.zonoid ....................................... 13
  depth.zonoid ............................................... 14
  is.in.convex ............................................... 16

Index 18
Description

The package implements the DDα-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: 1.0.5
Date: 2014-03-14
License: GPL-2

Use `ddalpha.train` to train the DD\( \alpha \)-classifier and `ddalpha.classify` to classify with it. Either the zonoid or Tukey depth can be used for the depth transformation. The zonoid depth is exactly computed by the function `depth.zonoid`, the Tukey depth is approximated by the function `depth.randomTukey`. Corresponding to them depth representations (`depth.space.zonoid` and `depth.space.randomTukey` for zonoid and the random Tukey depth respectively) are obtained, check by `is.in.convex` 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.

Author(s)

Oleksii Pokotylo, <alexy.pokotylo at gmail.com>

Pavlo Mozharovskyi, <mozharovskyi at statistik.uni-koeln.de>

References


Mozharovskyi, P., Mosler, K. and Lange, T. (2013), Classifying real-world data with the DD\( \alpha \)-procedure, Mimeo.
See Also

ddalpa.train, ddalpa.classify, depth.zonoid, depth.randomTukey, depth.space.zonoid, depth.space.randomTukey, is.in.convex.

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, 4), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2, 2),
                  matrix(c(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
ddalpa <- ddalpa.train(data$train)

ddalpa

# Classify by means of DDalpha-classifier
classes <- ddalpa.classify(data$test[, propertyVars], ddalpa)
cat("Classification error rate:",
    sum(unlist(classes) != data$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.randomTukey <- depth.randomTukey(data$test[1:10, propertyVars],
                                         data$train[trainIndices, propertyVars])
cat("Random Tukey depths:", depths.randomTukey, "\n")

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

# Calculate depth space with the random Tukey depth
dspace.randomTukey <- depth.space.randomTukey(data$train, c(100, 100))

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

Classify data using the DD_\alpha_-classifier and a specified outsider treatment.

**Usage**

```r
ddalpa.classify(objects, ddalpha, outsider.method = "LDA", use.convex = NULL)
```

**Arguments**

- **objects** Matrix containing objects to be classified; each row is one d-dimensional object.
- **ddalpha** DD_\alpha_-classifier (obtained by `ddalpha.train`).
- **outsider.method** Character string, name of a treatment to be used for outsiders; one of those trained by `ddalpha.train`. If the treatment was specified using the argument `outsider.methods` 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 depth = "zonoid" both values give the same result. If NULL the value specified in DD_\alpha_-classifier (in `ddalpha.train`) is used.

**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.

**Author(s)**

The algorithm for computation of zonoid depth (Dyckerhoff, Koshevoy and Mosler, 1996) has been implemented in C++ by Rainer Dyckerhoff.

**References**


ddalpha.train

See Also

ddalpha.train to train the DD\(\alpha\)-classifier.

Examples

```r
# 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(1,1,1,1), nrow = 2, ncol = 2, byrow = TRUE))
class2 <- mvrnorm(200, c(2, 2),
                  matrix(c(1,1,1,1), 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="")
```

Description

Trains the DD\(\alpha\)-classifier (Lange, Mosler and Mozharovskyi, 2014; Mozharovskyi, Mosler and Lange, 2013) using a training sample according to given parameters. The DD\(\alpha\)-classifier is a non-parametric procedure that first transforms the training sample into the depth space calculating for that depth of each point w.r.t each class (dimension of this space equals the number of classes in the training sample then), and then 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). 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 first too.
The current realization of the DDo-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(data, depth = "randomTukey", separator = "alpha", outsider.methods = "LDA", outsider.settings = NULL, aggregation.method = "majority", knnrange = NULL, numchunks = 10, numdirections = 1000, use.convex = FALSE, max.degree = 3, mah.estimate = "moment", mah.parmcd = 0.75, mah.priors = NULL)

Arguments

data Matrix containing training sample where each row is one object of the training sample where first $d$ entries are inputs and the last entry is output (class label).
depth Character string determining which depth notion to use; can be "randomTukey" (the default) or "zonoid".
separator The method used for separation of the DD-plot; can be "alpha" (the default) or "knnlm".
outsider.methods Vector of character strings each being a name of a basic outsider method for eventual classification; possible names are: "LDA" (the default), "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'.
outsider.settings 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.
aggregation.method Character string determining which method to apply to aggregate binary classification results during multiclass classification; can be "majority" (the default) or "sequent". If "majority", $q(q - 1)/2$ (with $q$ 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", q binary 'one against all'-classifiers are trained and ties during the classification are resolved as before.

knn.range The maximal number of neighbours for kNN separation.
	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".

num.directions Number of directions to use when calculating the random Tukey depth (i.e. when depth = "randomTukey"); should be > 1.

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.

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.

Details

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 \texttt{lda}; \( k \)-Nearest-Neighbor Classifier ("kNN"), see \texttt{knn, knn.cv}; Affine-Invariant kNN ("kNNAff"), an affine-invariant version of the kNN, 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:

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

The algorithm for computation of zonoid depth (Dyckerhoff, Koshevoy and Mosler, 1996) has been implemented in C++ by Rainer Dyckerhoff.

References


See Also

ddalphaNclassify for classification using DDα-classifier, depthNspaceNzonoid and depthNspaceNrandomTukey for calculation of depth spaces, isNinNconvex 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$train)
classes1 <- ddalpha.classify(data$test[,propertyVars], ddalpha1)
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", depth
Calculate the Random Tukey Depth

**Description**

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

**Usage**

```r
depth.randomTukey(x, data, num.directions = 1000)
```

**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.
- `num.directions` Number of random directions to be generated.
Details

Calculates Tukey (= halfspace, location) depth (Tukey, 1975) 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.

Value

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

References


See Also

`depth.zonoid` for calculation of zonoid depth.

Examples

```r
# 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.randomTukey(x, data)
cat("Depths: ", depths, "\n")
```
Calculate Depth Space using the Random Tukey Depth

**Description**
Calculates the depth space of the training sample using the random Tukey depth.

**Usage**

```
depth.space.randomTukey(data, cardinalities, num.directions = 1000)
```

**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.
- `num.directions`: Number of random directions to be generated.

**Details**

The depth representation is calculated in the same way as in `depth.randomTukey`, 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**


See Also

ddalpha.train and ddalpha.classify for application, depth.randomTukey for calculation of the random 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(2,2),
                matrix(c(1,1,1,4), nrow = 2, ncol = 2, byrow = TRUE))
data <- rbind(class1, class2)
# Get depth space using the random Tukey depth
depth.space.zonoid(data, c(10, 10))

---

depth.space.zonoid  Calculate Depth Space using Zonoid Depth

depth.space.zonoid  Calculate Depth Space using Zonoid Depth

depth.space.zonoid  Calculate Depth Space using Zonoid Depth

Description

Calculates the depth space of the training sample using zonoid depth.

Usage

depth.space.zonoid(data, cardinalities)

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.

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.
The algorithm for computation of zonoid depth (Dyckerhoff, Koshevoy and Mosler, 1996) has been implemented in C++ by Rainer Dyckerhoff.

References


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))

depth.zonoid Calculate Zonoid Depth

Description

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

Usage

depth.zonoid(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 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.

Author(s)

The algorithm for computation of zonoid depth (Dyckerhoff, Koshevoy and Mosler, 1996) has been implemented in C++ by Rainer Dyckerhoff.

References


See Also

depth.randomTukey for calculation of the random Tukey depth.

Examples

```r
# 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))
```
is.in.convex

Description

Checks the belonging to at least one of class convex hulls of the training sample.

Usage

is.in.convex(x, data, cardinalities)

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.

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.
is.in.convex

References


See Also
ddalpa.train and ddalpa.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(is.in.convex(data$test[,propertyVars],
                                 data$train[,propertyVars], c(200, 200)) == 0)
  cat(numOutsiders, "outsiders found in the testing sample.\n")
Index

*Topic **classif**
  - ddalpha-package, 2
  - ddalpha.classify, 4
  - ddalpha.train, 5

*Topic **multivariate**
  - ddalpha-package, 2
  - ddalpha.classify, 4
  - ddalpha.train, 5
  - depth.randomTukey, 10
  - depth.space.randomTukey, 12
  - depth.space.zonoid, 13
  - depth.zonoid, 14
  - is.in.convex, 16

*Topic **nonparametric**
  - ddalpha-package, 2
  - ddalpha.classify, 4
  - ddalpha.train, 5
  - depth.randomTukey, 10
  - depth.space.randomTukey, 12
  - depth.space.zonoid, 13
  - depth.zonoid, 14

*Topic **package**
  - ddalpha-package, 2

*Topic **robust**
  - ddalpha-package, 2
  - ddalpha.classify, 4
  - ddalpha.train, 5
  - depth.randomTukey, 10
  - depth.space.randomTukey, 12
  - depth.space.zonoid, 13
  - depth.zonoid, 14

covMcd, 8

ddalpa (ddalpha-package), 2
ddalpha-package, 2
ddalpha.classify, 2, 3, 4, 9, 13, 14, 17
ddalpha.train, 2–5, 5, 13, 14, 17
depth.randomTukey, 2, 3, 10, 12, 13, 15
depth.space.randomTukey, 2, 3, 9, 12

depth.space.zonoid, 2, 3, 9, 13
depth.zonoid, 2, 3, 11, 13, 14, 14
is.in.convex, 2, 3, 9, 16
knn, 7
knn.cv, 7
lda, 7

18