Package ‘classiFunc’

April 16, 2018

Type Package

Title Classification of Functional Data

Version 0.1.1

Date 2018-03-29

URL `https://github.com/maierhofert/classiFunc`

Description Efficient implementation of k-
nearest neighbor estimation and kernel estimation for functional data classification.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Depends R (>= 2.10)

Suggests testthat, knitr, rmarkdown, parallelMap

Imports BBmisc (>= 1.11), checkmate (>= 1.8.2), dtw, fda, fda.usc,
fdasrvf, proxy, rucrdtw, stats, zoo

VignetteBuilder knitr

NeedsCompilation no

Author Thomas Maierhofer [aut, cre],
Karen Fuchs [ctb],
Florian Pfisterer [aut]

Maintainer Thomas Maierhofer <`thomasjmaierhofer@gmail.com`>

Repository CRAN

Date/Publication 2018-04-16 17:15:34 UTC

R topics documented:

`ArrowHead` *The shape of arrow heads.*

Description

A dataset containing the outline of arrow heads as functional covariable. This is a subset of the "ArrowHead" data set of the UCR TSC repository.

Usage

```
ArrowHead
```

Format

An object of class `data.frame` with 100 rows and 84 columns.

Details

The arrowhead data consists of outlines of the images of arrowheads. The shapes of the projectile points are converted into a time series using the angle-based method. The classification of projectile points is an important topic in anthropology. The classes are based on shape distinctions such as the presence and location of a notch in the arrow. The problem in the repository is a length normalized version of that used in Ye09shapelets. The three classes are called "Avonlea", "Clovis" and "Mix".

Format A data frame with 100 rows (=observations) and 84 variables

col 1:83 shape of a projectile as functional observation.

target encoding the class of the projectile.

Source

http://timeseriesclassification.com/description.php?Dataset=ArrowHead

`BeetleFly` *Beetle/Fly Data*

Description

Classification of Beetle and Fly outlines.

Usage

```
BeetleFly
```

Format

An object of class `data.frame` with 40 rows and 513 columns.

Details

MPEG-7 CE Shape-1 Part B is a database of binary images developed for testing MPEG-7 shape descriptors, and is available free online (http://www.dabi.temple.edu/~shape/MPEG7/dataset.html). It is used for testing contour/image and skeleton-based descriptors. Classes of images vary broadly, and include classes that are similar in shape to one another. There are 20 instances of each class, and 60 classes in total. We have extracted the outlines of these images and mapped them into 1-D series of distances to the centre. Beetle/Fly is the problem of distinguishing between an outline of a beetle and a fly

Format A data frame made up of

att0 to att511 Functional observation of outline.

target Factor encoding if outline is from a beetle (1) or a fly (2).

References

Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2014). "Classification of time series by shapelet transformation." Data Mining and Knowledge Discovery, 28(4), 851-881. URL http://timeseriesclassification.com/descriptio

`classiFunc` *The classiFunc package*

Description

This package implements methods for functional data classification. The main functions of this package are `classiKnn`, a k nearest neighbor estimator for functional data, and `classiKernel`, a kernel estimator for functional data. The package uses efficiently implemented semimetrics to create the distance matrix of the functional observations in the function `computeDistMat`. Currently supported distance measures are all `methods` implemented in `dist` and all semimetrics suggested in Fuchs et al. (2015). Additionally, all (semi-)metrics can be used on a derivative of arbitrary order of the functional observations. This is a new package, please report all bugs and issues at `https://github.com/maierhofert/classiFunc`.

Author(s)

Thomas Maierhofer Florian Pfisterer

References

Fuchs, K., J. Gertheiss, and G. Tutz (2015): Nearest neighbor ensembles for functional data with interpretable feature selection. Chemometrics and Intelligent Laboratory Systems 146, 186 - 197.

`classiKernel` *Create a kernel estimator for functional data classification*

Description

Creates an efficient kernel estimator for functional data classification. Currently supported distance measures are all `metrics` implemented in `dist` and all semimetrics suggested in Fuchs et al. (2015). Additionally, all (semi-)metrics can be used on a derivative of arbitrary order of the functional observations. For kernel functions all kernels implemented in `fda.usc` are available as well as custom kernel functions.

Usage

```
classiKernel(classes, fdata, grid = 1:ncol(fdata), h = 1, metric = "L2",
  ker = "Ker.norm", nderiv = 0L, derived = FALSE,
  deriv.method = "base.diff", custom.metric = function(x, y, ...) {
  return(sqrt(sum((x - y)^2))) }, custom.ker = function(u) {
  return(dnorm(u)) }, ...)
```

Arguments

`classes`	[`factor(nrow(fdata))`] factor of length `nrow(fdata)` containing the classes of the observations.
`fdata`	[`matrix`] matrix containing the functional observations as rows.
`grid`	[`numeric(ncol(fdata))`] numeric vector of length `ncol(fdata)` containing the grid on which the functional observations were evaluated.
`h`	[numeric(1)] controls the bandwidth of the kernel function. All kernel functions `ker` should be implemented to have bandwidth = 1. The bandwidth is controlled via `h` by using `K(x) = ker(x/h)` as the kernel function.
`metric`	[`character(1)`] character string specifying the (semi-)metric to be used. For a an overview of what is available see the `method` argument in `computeDistMat`. For a full list execute `metricChoices()`.

`ker`	[numeric(1)] character string describing the kernel function to use. Available are amongst others all kernel functions from `Kernel`. For the full list execute `kerChoices()`. The usage of customized kernel function is symbolized by `ker = "custom.ker"`. The customized function can be specified in `custom.ker`
`nderiv`	`[integer(1)]` order of derivation on which the metric shall be computed. The default is 0L.
`derived`	`[logical(1)]` Is the data given in `fdata` already derived? Default is set to `FALSE`, which will lead to numerical derivation if `nderiv >= 1L` by applying `deriv.fd` on a `Data2fd` representation of `fdata`.
`deriv.method`	`[character(1)]` character indicate which method should be used for derivation. Currently implemented are `"base.diff"`, the default, and `"fda.deriv.fd"`. `"base.diff"` uses the method `base::diff` for equidistant measures without missing values, which is faster than transforming the data into the class `fd` and deriving this using `fda::deriv.fd`. The second variant implies smoothing, which can be preferable for calculating high order derivatives.
`custom.metric`	`[function(x, y, ...)]` only used if `deriv.method = "custom.method"`. A function of functional observations `x` and `y` returning their distance. The default is the L2 distance. See how to implement your distance function in `dist`.
`custom.ker`	[function(u)] customized kernel function. This has to be a function with exactly one parameter `u`, returning the numeric value of the kernel function `ker(u)`. This function is only used if `ker == "custom.ker"`. The bandwidth should be constantly equal to 1 and is controlled via `h`.
`...`	further arguments to and from other methods. Hand over additional arguments to `computeDistMat`, usually additional arguments for the specified (semi-)metric. Also, if `deriv.method == "fda.deriv.fd"` or `fdata` is not observed on a regular grid, additional arguments to `fdataTransform` can be specified which will be passed on to `Data2fd`.

Value

`classiKernel` returns an object of class `'classiKernel'`. This is a list containing at least the following components:

`classes` a factor of length nrow(fdata) coding the response of the training data set.

`fdata` the raw functional data as a matrix with the individual observations as rows.

`proc.fdata` the preprocessed data (missing values interpolated, derived and evenly spaced). This data is `this.fdataTransform(fdata)`. See `this.fdataTransform` for more details.

`grid` numeric vector containing the grid on which `fdata` is observed)

`h` numeric value giving the bandwidth to be used in the kernel function.

`ker` character encoding the kernel function to use.

`metric` character string coding the distance metric to be used in `computeDistMat`.

`nderiv` integer giving the order of derivation that is applied to fdata before computing the distances between the observations.

`this.fdataTransform` preprocessing function taking new data as a matrix. It is used to transform `fdata` into `proc.fdata` and is required to preprocess new data in order to predict it. This function ensures, that preprocessing (derivation, respacing and interpolation of missing values) is done in the exact same way for the original training data set and future (test) data sets.

`call` the original function call.

References

Fuchs, K., J. Gertheiss, and G. Tutz (2015): Nearest neighbor ensembles for functional data with interpretable feature selection. Chemometrics and Intelligent Laboratory Systems 146, 186 - 197.

See Also

predict.classiKernel

Examples

```
# How to implement your own kernel function
data("ArrowHead")
classes = ArrowHead[,"target"]

set.seed(123)
train_inds = sample(1:nrow(ArrowHead), size = 0.8 * nrow(ArrowHead), replace = FALSE)
test_inds = (1:nrow(ArrowHead))[!(1:nrow(ArrowHead)) %in% train_inds]

ArrowHead = ArrowHead[,!colnames(ArrowHead) == "target"]

# custom kernel
myTriangularKernel = function(u) {
  return((1 - abs(u)) * (abs(u) < 1))
}

# create the model
mod1 = classiKernel(classes = classes[train_inds], fdata = ArrowHead[train_inds,],
                    ker = "custom.ker", h = 2, custom.ker = myTriangularKernel)

# calculate the model predictions
pred1 = predict(mod1, newdata = ArrowHead[test_inds,], predict.type = "response")

# prediction accuracy
mean(pred1 == classes[test_inds])

# create another model using an existing kernel function
mod2 = classiKernel(classes = classes[train_inds], fdata = ArrowHead[train_inds,],
                    ker = "Ker.tri", h = 2)

# calculate the model predictions
pred2 = predict(mod1, newdata = ArrowHead[test_inds,], predict.type = "response")
```

```
# prediction accuracy
mean(pred2 == classes[test_inds])
## Not run:
# Parallelize across 2 CPU's
library(parallelMap)
parallelStartSocket(2L) # parallelStartMulticore for Linux
predict(mod1, newdata =  fdata[test_inds,], predict.type = "prob", parallel = TRUE, batches = 2L)
parallelStop()

## End(Not run)
```

---

classiKnn *Create a knn estimator for functional data classification.*

Description

Creates an efficient k nearest neighbor estimator for functional data classification. Currently supported distance measures are all `metrics` implemented in `dist` and all semimetrics suggested in Fuchs et al. (2015). Additionally, all (semi-)metrics can be used on an arbitrary order of derivation.

Usage

```
classiKnn(classes, fdata, grid = 1:ncol(fdata), knn = 1L, metric = "L2",
  nderiv = 0L, derived = FALSE, deriv.method = "base.diff",
  custom.metric = function(x, y, ...) {     return(sqrt(sum((x - y)^2))) },
  ...)
```

Arguments

`classes`	`[factor(nrow(fdata))]` factor of length `nrow(fdata)` containing the classes of the observations.
`fdata`	`[matrix]` matrix containing the functional observations as rows.
`grid`	`[numeric(ncol(fdata))]` numeric vector of length `ncol(fdata)` containing the grid on which the functional observations were evaluated.
`knn`	`[integer(1)]` number of nearest neighbors to use in the k nearest neighbor algorithm.
`metric`	`[character(1)]` character string specifying the (semi-)metric to be used. For a an overview of what is available see the `method` argument in `computeDistMat`. For a full list execute `metricChoices()`.
`nderiv`	`[integer(1)]` order of derivation on which the metric shall be computed. The default is 0L.

`derived`	`[logical(1)]` Is the data given in `fdata` already derived? Default is set to `FALSE`, which will lead to numerical derivation if `nderiv >= 1L` by applying `deriv.fd` on a `Data2fd` representation of `fdata`.
`deriv.method`	`[character(1)]` character indicate which method should be used for derivation. Currently implemented are `"base.diff"`, the default, and `"fda.deriv.fd"`. `"base.diff"` uses the method `base::diff` for equidistant measures without missing values, which is faster than transforming the data into the class `fd` and deriving this using `fda::deriv.fd`. The second variant implies smoothing, which can be preferable for calculating high order derivatives.
`custom.metric`	`[function(x, y, ...)]` only used if `deriv.method = "custom.method"`. A function of functional observations `x` and `y` returning their distance. The default is the L2 distance. See how to implement your distance function in `dist`.
`...`	further arguments to and from other methods. Hand over additional arguments to `computeDistMat`, usually additional arguments for the specified (semi-)metric. Also, if `deriv.method == "fda.deriv.fd"` or `fdata` is not observed on a regular grid, additional arguments to `fdataTransform` can be specified which will be passed on to `Data2fd`.

Value

`classiKnn` returns an object of class `"classiKnn"`. This is a list containing at least the following components:

`call` the original function call.

`classes` a factor of length nrow(fdata) coding the response of the training data set.

`fdata` the raw functional data as a matrix with the individual observations as rows.

`grid` numeric vector containing the grid on which `fdata` is observed)

`proc.fdata` the preprocessed data (missing values interpolated, derived and evenly spaced). This data is `this.fdataTransform(fdata)`. See `this.fdataTransform` for more details.

`knn` integer coding the number of nearest neighbors used in the k nearest neighbor classification algorithm.

`metric` character string coding the distance metric to be used in `computeDistMat`.

`nderiv` integer giving the order of derivation that is applied to fdata before computing the distances between the observations.

`this.fdataTransform` preprocessing function taking new data as a matrix. It is used to transform `fdata` into `proc.fdata` and is required to preprocess new data in order to predict it. This function ensures, that preprocessing (derivation, respacing and interpolation of missing values) is done in the exact same way for the original training data set and future (test) data sets.

References

Fuchs, K., J. Gertheiss, and G. Tutz (2015): Nearest neighbor ensembles for functional data with interpretable feature selection. Chemometrics and Intelligent Laboratory Systems 146, 186 - 197.

See Also

predict.classiKnn

Examples

```
# Classification of the Phoneme data
data(Phoneme)
classes = Phoneme[,"target"]

set.seed(123)
# Use 80% of data as training set and 20% as test set
train_inds = sample(1:nrow(Phoneme), size = 0.8 * nrow(Phoneme), replace = FALSE)
test_inds = (1:nrow(Phoneme))[!(1:nrow(Phoneme)) %in% train_inds]

# create functional data as matrix with observations as rows
fdata = Phoneme[,!colnames(Phoneme) == "target"]

# create k = 3 nearest neighbors classifier with L2 distance (default) of the
# first order derivative of the data
mod = classiKnn(classes = classes[train_inds], fdata = fdata[train_inds,],
                 nderiv = 1L, knn = 3L)

# predict the model for the test set
pred = predict(mod, newdata =  fdata[test_inds,], predict.type = "prob")

## Not run:
# Parallelize across 2 CPU's
library(parallelMap)
parallelStartSocket(cpus = 2L) # parallelStartMulticore(cpus = 2L) for Linux
predict(mod, newdata =  fdata[test_inds,], predict.type = "prob", parallel = TRUE, batches = 2L)
parallelStop()

## End(Not run)
```

computeDistMat *Compute a distance matrix for functional observations*

Description

This mainly internal function offers a unified framework to access the `dist` function from the `proxy` package and additional (semi-)metrics.

Usage

```
computeDistMat(x, y = NULL, method = "Euclidean", dmin = 0, dmax = 1,
  dmin1 = 0, dmax1 = 1, dmin2 = 0, dmax2 = 1, t1 = 0, t2 = 1,
  .poi = seq(0, 1, length.out = ncol(x)), custom.metric = function(x, y, lp
  = 2, ...) {     return(sum(abs(x - y)^lp)^(1/lp)) }, a = NULL, b = NULL,
  c = NULL, lambda = 0, ...)
```

Arguments

x	[matrix] matrix containing the functional observations as rows.
y	[matrix] see x. The default NULL uses y = x.
method	[character(1)] character string describing the distance function to be used. For a full list execute metricChoices().

Euclidean equals Lp with p = 2. This is the default.

Lp, Minkowski the distance for an Lp-space, takes p as an additional argument in ....

Manhattan equals Lp with p = 1.

supremum, max, maximum equals Lp with p = Inf. The supremal pointwise difference between the curves.

and ... all other available measures for dist.

shortEuclidean Euclidean distance on a limited part of the domain. Additional arguments dmin and dmax can be specified in ..., giving the position of the first and the last point to use of an evenly spaced sequence from 0 to 1 of length length(grid). The default values are dmin = o and dmax = 1, which results in the Euclidean distance on the entire domain.

mean the absolute similarity of the overall mean values of the observations.

relAreas the difference of the relation of two areas on parts of the domain given by dmin1 to dmax1 and dmin2 to dmax2. They are defined analogously to dmin and dmax and take the same default values.

jump the similarity of jump heights at points t1 and t2, i.e. x[t1 * length(x)] - x[t2 * length(x)] for every functional observation x. The points t1 and t2 are the positions in an evenly spaced sequence from 0 to 1 of length length(grid) for which to compare the jump height. The default values are t1 = 0 and t2 = 1.

globMax the difference of the curves global maxima.

globMin the difference of the curves global minima.

points the mean absolute differences at certain observation points .poi, also called "points of impact". These are specified as a vector .poi of arbitrary length with values between 0 and 1, encoding the the index of the points of observations. The default value is .poi = seq(0, 1, length.out = length(grid)), which results in the Manhattan distance.

custom.metric your own semimetric will be used. Specify your own distance function in the argument custom.metric.

amplitudeDistance,phaseDistance The amplitude distance or phase distance as described in Srivastava, A. and E. P. Klassen (2016). Functional and Shape Data Analysis. Springer.

FisherRao, elasticMetric the elastic distance of the square root velocity of the curves as described in Srivastava and Klassen (2016). This equates to the Fisher Rao metric.

elasticDistance weighted mean of the amplitude and the phase distance using the implementation in elastic.distance. Additional arguments are the numeric the penalization parameters `a,b,c` for the amplitude distance (`a^2`) and the phase distance (`b^2`). The default values are `a = 1/2, b = 1`. Alternatively `c` denotes the ratio of `2*a` and `b`. `lambda` is the additional penalization parameter for the warping allowed before calculating the elastic distance. The default is 1.

`rucrdtw, rucred` Dynamic Time Warping Distance and Euclidean Distance from package `rucrdtw`. Implemented in Boersch-Supan (2016) and originally described in Rakthanmanon et al. (2012).

`dmin, dmax, dmin1, dmax1, dmin2, dmax2`
: `[integer(1)]`
encode the indices used to define subspaces for `method %in% c("shortEuclidean", "relAreas")` as numeric values between 0 and 1, where 0 encodes `grid[1]` and 1 encodes `grid[length(grid)]`.

`t1, t2`
: `[numeric(1)]`
encode the position of the points for which to compare the jump heights in `method = "jump"` as numeric values between 0 and 1, see `dmin`.

`.poi`
: `[numeric(1 to ncol(x))]`
numeric vector of length arbitrary length taking numeric values between 0 and 1, denoting the position of the points of interest for `method = "points"`. The default value is `.poi = seq(0, 1, length.out = length(grid))`, which results in the Manhattan distance.

`custom.metric`
: `[function(x, y, ...)]`
a function specifying how to compute the distance between two functional observations (= numeric vectors of the same length) `x` and `y`. It can handle additional arguments in `...`. The default is the Euclidean distance (equals Minkwoski distance with `lp = 2`). Used for `method = "custom.metric"`.

`a, b, c`
: `[numeric(1)]`
weights of the amplitude distance (`a`) and the phase distance (`b`) in a semimetric that combines them by addition. Used for `method == 'elasticDistance'`.

`lambda`
: `[numeric(1)]`
penalization parameter for the warping allowed before calculating the elastic distance. Default value is 0. Large values imply less (no) warping, small values imply more warping. Used for `method %in% c('elastic', 'SRV')`.

`...`
: additional parameters to the (semi-)metrics.

Value

a matrix of dimensions `nrow(x)` by `nrow(y)` containing the distances of the functional observations contained in `x` and `y`, if `y` is specified. Otherwise a matrix containing the distances of all functional observations within `x` to each other.

References

Boersch-Supan (2016). rucrdtw: Fast time series subsequence search in R. The Journal of Open Source Software URL http://doi.org/10.21105/joss.00100

Fuchs, K., J. Gertheiss, and G. Tutz (2015): Nearest neighbor ensembles for functional data with interpretable feature selection. Chemometrics and Intelligent Laboratory Systems 146, 186 - 197.

Rakthanmanon, Thanawin, et al. "Searching and mining trillions of time series subsequences under dynamic time warping." Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012.

Srivastava, A. and E. P. Klassen (2016). Functional and Shape Data Analysis. Springer.

DTI *Diffusion Tensor Imaging: tract profiles and outcomes*

Description

Fractional anisotropy (FA) tract profiles for the corpus callosum (cca) and the right corticospinal tract (rcst). Accompanying the tract profiles are the subject ID numbers, visit number, total number of scans, multiple sclerosis case status and Paced Auditory Serial Addition Test (pasat) score.

Format

A data frame made up of

cca A 382 x 93 matrix of fractional anisotropy tract profiles from the corpus callosum. Missing values were imputed using splines;

rcst A 382 x 55 matrix of fractional anisotropy tract profiles from the right corticospinal tract. Missing values were imputed using splines;

ID Numeric vector of subject ID numbers;

visit Numeric vector of the subject-specific visit numbers;

visit.time Numeric vector of the subject-specific visit time, measured in days since first visit;

Nscans Numeric vector indicating the total number of visits for each subject;

case Numeric vector of multiple sclerosis case status: 0 - healthy control, 1 - MS case;

sex factor variable indicated subject's sex;

pasat Numeric vector containing the PASAT score at each visit.

Details

If you use this data as an example in written work, please include the following acknowledgment: “The MRI/DTI data were collected at Johns Hopkins University and the Kennedy-Krieger Institute"

Data and description was copied from the `refund` package.

References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized Functional Regression. *Journal of Computational and Graphical Statistics*, 20, 830 - 851.

Goldsmith, J., Crainiceanu, C., Caffo, B., and Reich, D. (2010). Longitudinal Penalized Functional Regression for Cognitive Outcomes on Neuronal Tract Measurements. *Journal of the Royal Statistical Society: Series C*, 61, 453 - 469.

`DTI_original` *Diffusion Tensor Imaging: tract profiles and outcomes*

Description

Fractional anisotropy (FA) tract profiles for the corpus callosum (cca) and the right corticospinal tract (rcst). Accompanying the tract profiles are the subject ID numbers, visit number, total number of scans, multiple sclerosis case status and Paced Auditory Serial Addition Test (pasat) score.

Format

A data frame made up of

cca A 382 x 93 matrix of fractional anisotropy tract profiles from the corpus callosum containing missing values;

rcst A 382 x 55 matrix of fractional anisotropy tract profiles from the right corticospinal tract containing missing values;

ID Numeric vector of subject ID numbers;

visit Numeric vector of the subject-specific visit numbers;

visit.time Numeric vector of the subject-specific visit time, measured in days since first visit;

Nscans Numeric vector indicating the total number of visits for each subject;

case Numeric vector of multiple sclerosis case status: 0 - healthy control, 1 - MS case;

sex factor variable indicated subject's sex;

pasat Numeric vector containing the PASAT score at each visit.

Details

If you use this data as an example in written work, please include the following acknowledgment: "The MRI/DTI data were collected at Johns Hopkins University and the Kennedy-Krieger Institute"

Data and description was copied from the `refund` package.

References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized Functional Regression. *Journal of Computational and Graphical Statistics*, 20, 830 - 851.

Goldsmith, J., Crainiceanu, C., Caffo, B., and Reich, D. (2010). Longitudinal Penalized Functional Regression for Cognitive Outcomes on Neuronal Tract Measurements. *Journal of the Royal Statistical Society: Series C*, 61, 453 - 469.

`fdataTransform` *Create a preprocessing pipeline function*

Description

Internal function, documented due to the importance of its concept. Creates a pipeline function to do all the preprocessing needed in `classiKnn` and `classiKernel`. This is helpful to ensure that the data preprocessing (imputation of missing values, derivation) is carried out in exactly the same way for the training and the test set in `predict.classiKnn` and `predict.classiKernel`.

Usage

```
fdataTransform(grid, nderiv, derived, evenly.spaced, no.missing, deriv.method,
  ...)
```

Arguments

`grid, nderiv, derived, evenly.spaced, no.missing, deriv.method`	see `classiKnn`
`...`	additional arguments to fda::smooth.basis

Value

Pipeline function taking one argument `fdata`. The returned function carries out all the preprocessing needed for the calling model of class `classiKnn`.

`Growth` *Berkeley Growth Study Data (regular grid)*

Description

A data frame containing the heights of 39 boys and 54 girls from age 1 to 18, the ages at which they were collected.

Format

A list made up of

ID Factor of length 93 containing the subject IDs

sex Factor encoding the sex of children with values in `` c(`male`, `female`) ``

height A 93 x 31 matrix giving the height in cm of 93 children at 31 ages

@inheritSection Growth_irregular references @seealso Growth_irregular

Details

Data and description was reformatted from the `fda` package to be observed on a regular grid in one year steps.

`Growth_irregular` *Berkeley Growth Study Data*

Description

A data frame containing the heights of 39 boys and 54 girls from age 1 to 18, the ages at which they were collected.

Format

A list made up of

ID Factor of length 93 containing the subject IDs

sex Factor encoding the sex of children with values in `` c(`male`, `female`) ``

age Numeric vector of length 31 encoding the age at the measurements

height A 93 x 31 matrix giving the height in cm of 93 children at 31 ages

Details

Data and description was reformatted from the `fda` package.

The ages are not equally spaced, see `Growth$age`.

References

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

Growth

`kerChoices` *List the names of all implemented kernel functions*

Description

`kerChoices` is a function returning the names of all kernel functions that are currently implemented in the `classiFunc`-package and can be used for the argument `ker` in `classiKernel`.

Usage

```
kerChoices()
```

metricChoices — *List the names of all metrics*

Description

`metricChoices` is a function returning the names of all (semi-)metrics that are currently implemented in the `link{classiFunc}`-package and can be used for the argument `method` in `computeDistMat` or the argument `metric` in `classiKnn` and `classiKernel` respectively.

Usage

```
metricChoices(proxy.only = FALSE)
```

Arguments

`proxy.only`	[logical(1)] should only the metrics of the proxy package be returned? Defaults to `FALSE`, which results in returning the names of all allowed metrics for `computeDistMat`.

parallelComputeDistMat — *Paralleize computing a distance matrix for functional observations*

Description

Uses `parallelMap` to parallelize the computation of the distance matrix. This is done by dividing the data into batches and computing the distance matrix for each batch. For details on distance computation see `computeDistMat`.

Usage

```
parallelComputeDistMat(x, y = NULL, method = "Euclidean", batches = 1L,
  ...)
```

Arguments

`x`	`[matrix]` matrix containing the functional observations as rows.
`y`	`[matrix]` see `x`. The default `NULL` uses `y = x`.
`method`	`[character(1)]` character string describing the distance function to be used. For a full list execute `metricChoices()`. `Euclidean` equals `Lp` with `p = 2`. This is the default.

`Lp, Minkowski` the distance for an Lp-space, takes `p` as an additional argument in `...`.

`Manhattan` equals `Lp` with `p = 1`.

`supremum, max, maximum` equals `Lp` with `p = Inf`. The supremal pointwise difference between the curves.

`and ...` all other available measures for `dist`.

`shortEuclidean` Euclidean distance on a limited part of the domain. Additional arguments `dmin` and `dmax` can be specified in `...`, giving the position of the first and the last point to use of an evenly spaced sequence from `0` to `1` of length `length(grid)`. The default values are `dmin = o` and `dmax = 1`, which results in the Euclidean distance on the entire domain.

`mean` the absolute similarity of the overall mean values of the observations.

`relAreas` the difference of the relation of two areas on parts of the domain given by `dmin1` to `dmax1` and `dmin2` to `dmax2`. They are defined analogously to `dmin` and `dmax` and take the same default values.

`jump` the similarity of jump heights at points `t1` and `t2`, i.e. `x[t1 * length(x)] - x[t2 * length(x)]` for every functional observation `x`. The points `t1` and `t2` are the positions in an evenly spaced sequence from `0` to `1` of length `length(grid)` for which to compare the jump height. The default values are `t1 = 0` and `t2 = 1`.

`globMax` the difference of the curves global maxima.

`globMin` the difference of the curves global minima.

`points` the mean absolute differences at certain observation points `.poi`, also called "points of impact". These are specified as a vector `.poi` of arbitrary length with values between `0` and `1`, encoding the the index of the points of observations. The default value is `.poi = seq(0, 1, length.out = length(grid))`, which results in the Manhattan distance.

`custom.metric` your own semimetric will be used. Specify your own distance function in the argument `custom.metric`.

`amplitudeDistance,phaseDistance` The amplitude distance or phase distance as described in Srivastava, A. and E. P. Klassen (2016). Functional and Shape Data Analysis. Springer.

`FisherRao, elasticMetric` the elastic distance of the square root velocity of the curves as described in Srivastava and Klassen (2016). This equates to the Fisher Rao metric.

`elasticDistance` weighted mean of the amplitude and the phase distance using the implementation in `elastic.distance`. Additional arguments are the numeric the penalization parameters `a,b,c` for the amplitude distance (`a^2`) and the phase distance (`b^2`). The default values are `a = 1/2, b = 1`. Alternatively `c` denotes the ratio of `2*a` and `b`. `lambda` is the additional penalization parameter for the warping allowed before calculating the elastic distance. The default is 1.

`rucrdtw, rucred` Dynamic Time Warping Distance and Euclidean Distance from package `rucrdtw`. Implemented in Boersch-Supan (2016) and originally described in Rakthanmanon et al. (2012).

`batches`	`[integer(1)]` Number of roughly equal-sized batches to split data into. The distance computation is then carried out for each batch.

`...`	additional parameters to the (semi-)metrics.

Value

a matrix of dimensions `nrow(x)` by `nrow(y)` containing the distances of the functional observations contained in `x` and `y`, if `y` is specified. Otherwise a matrix containing the distances of all functional observations within `x` to each other.

`Phoneme` *Phonetic Time Series.*

Description

A data set containing the audio files of English words.

Usage

```
Phoneme
```

Format

An object of class `data.frame` with 100 rows and 65 columns.

Details

This data set is a subsample of the data used in Hamooni and Mueen (2014). Each series is extracted from the segmented audio collected from Google Translate, oxforddictionaries.com and the Merrriam-Webster online dictionary. Each of these sources have different features. Audio files collected from Google translate, Oxford, and Merrriam-Webster dictionaries are recorded at 22050, 44100 and 11025 samples per second respectively. All of them have male and female speakers in different ratios. The Oxford dictionary includes British and American accent pronunciation for each word. After data collection, they segment waveforms of the words to generate phonemes using the Forced Aligner tool from the Penn Phonetics Laboratory.

Format A data frame with 100 rows (=observations) and 65 variables

col 1:64 one functional observation.

target encoding the word of the functional observation

Source

http://timeseriesclassification.com/description.php?Dataset=Phoneme

References

Hamooni, Hossein, and Mueen, Abdullah. "Dual-domain hierarchical classification of phonetic time series." Data Mining (ICDM), 2014 IEEE International Conference on. IEEE, 2014.

`predict.classiKernel` *predict a classiKernel object*

Description

predict function for a classiKnn object.

Usage

```
## S3 method for class 'classiKernel'
predict(object, newdata = NULL,
  predict.type = "response", parallel = FALSE, ...)
```

Arguments

`object`	`[classiKernel]` object of class classiKernel to get predictions from
`newdata`	`[data.frame]` (optional) new data to predict from with observations as rows. Do not derive this data, this will be done automatically if required by the model. If `NULL`, the training data is predicted, currently without using a leave-one-out prediction.
`predict.type`	`[character(1)]` one of 'response' or 'prob', indicating the type of prediction. Choose 'response' to return a vector of length `nrow(newdata)` containing the most predicted class. Choose 'prob' to return a matrix with `nrow(newdata)` rows containing the probabilities for the classes as columns.
`parallel`	`[logical(1)]` Should the prediction be parallelized? Uses `parallelMap` for parallelization. See `...` for further arguments.
`...`	`[list]` additional arguments to computeDistMat.

See Also

classiKernel

`predict.classiKnn` *predict a classiKnn object*

Description

predict function for a classiKnn object.

Usage

```
## S3 method for class 'classiKnn'
predict(object, newdata = NULL,
  predict.type = "response", parallel = FALSE, ...)
```

Arguments

`object`	`[classiKnn]` object of class classiKnn to get predictions from
`newdata`	`[data.frame]` (optional) new data to predict from with observations as rows. Do not derive this data, this will be done automatically if required by the model. If `NULL`, the training data is predicted, currently without using a leave-one-out prediction.
`predict.type`	`[character(1)]` one of 'response' or 'prob', indicating the type of prediction. Choose 'response' to return a vector of length `nrow(newdata)` containing the most predicted class. Choose 'prob' to return a matrix with `nrow(newdata)` rows containing the probabilities for the classes as columns.
`parallel`	`[logical(1)]` Should the prediction be parallelized? Uses `parallelMap` for parallelization. See `...` for further arguments.
`...`	`[list]` additional arguments to computeDistMat.

See Also

classiKnn

Index