Package 'marmalaid'

August 30, 2023

Type Package

Title MARine MAchine Learning AID -
a collection of functions for utilising machine learning in marine biological science

Version 0.1

Date 2023-08-18

Description This package is a collection of tools to derive features from gridded spatio-temporal data (EOF & SOM analysis) for use in a machine learning framework including feature selection via NSGA-II.

License GPL-3

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 3.6.0),
ecr (>= 2.0.0)

Imports caret,
data.table,
doParallel,
dplyr,
foreach,
grDevices,
kohonen (>= 3.0.0),
maps,
parallel,
parallelMap (>= 1.5.0),
pracma,
ranger,
raster,
R.cache (>= 0.14.0),
RColorBrewer,
scales,
sp,
terra

R topics documented:

`calc.lags` *Calculate (multiple) Lags of a time series*

Description

Function to calculate lagged versions of time-series.

Usage

```
calc.lags(ts, lags)
```

Arguments

`ts`	A vector of a time series.
`lags`	A vector of lags (positive or negative).

Value

A `data.frame` containing the ts at different lags of the same length as the original ts. NAs are used to fill in the missing values resulting from the shift.

`calc.mean.over.Month` *Seasonal mean for spatio-temporal field*

Description

Function to calculate the mean over subsequent month in a raster brick.

Usage

```
calc.mean.over.Month(raster, time, month, shiftYear = TRUE, verbose = T)
```

Arguments

`raster`	A raster of class `RasterBrick` ('raster'-pkg) or `SpatRaster` ('terra'-pkg) with each layer corresponding to a time step.
`time`	A vector of dates corresponding to each layer in the raster `brick`.
`month`	A vector of numbers between 1 and 12 denoting the month over which to calculate the average (e.g. 3,4,5). Only subsequent month are allowed (e.g `c(12,1,2)` works; `c(1,3,5)` does not!).
`shiftYear`	A logical value. How to handle averages over subsequent month but different years (e.g. Dec.-Jan.-Feb.). If `TRUE` the time returned corresponds to the older year (e.g. 1989), otherwise to the younger one (1990).
`verbose`	A logical value. Print output to console?

Value

A raster of class `RasterBrick` ('raster'-pkg) or `SpatRaster` ('terra'-pkg) containing the avg. spatial fields per specified monthly period per year.

Examples

```
## For an artificial toy dataset
library(raster)
library(terra)
r = raster::raster(extent(c(-5,5,40,60)),
          res = c(0.1,0.1))
date = seq(as.Date("1990-01-01"),
          as.Date("2000-12-31"),by = "month")
br = raster::brick(lapply(seq_along(date), function(i) setValues(r,runif(ncell(r)))))
calc.mean.over.Month(raster = br, time = date,month = c(12,1,2),shiftYear = TRUE)
calc.mean.over.Month(raster = br, time = date,month = c(12,1,2),shiftYear = FALSE)
calc.mean.over.Month(raster = terra::rast(br), time = date,month = c(12,1,2),shiftYear = FALSE)
## For SST of the North Sea
data(sst.ahoi)
SST.winter = calc.mean.over.Month(raster = sst.ahoi,
                        time = as.Date(sub("X","",names(sst.ahoi)),format = "%Y.%m.%d"),
                                  month = c(12,1,2),shiftYear = TRUE)
plot(SST.winter,1:6)
```

`EOF.Monte.Carlo` *Finds Nr. of PCs to retain by means of Monte-Carlo analysis*

Description

A form of Horns parallel analysis to compare the eigenvalues of a PCA with those of a null-model (randomly permuted colums).

Usage

```
EOF.Monte.Carlo(
  raster,
  center = TRUE,
  scale = FALSE,
  size = 100,
  CI.size = 1000,
  plot = TRUE
)
```

Arguments

`raster`	A raster brick from which to calculate the spatial-PCA.
`center`	If the mean should be removed prior to analysis.
`scale`	If each grid point should be scaled by its variance.
`size`	Nr. of permutations to generate for the Null-model.
`CI.size`	Nr. of bootstrap samples for calculating confidence-intervals.
`plot`	A logical value, indicating if the eigenvalues of the Null-model, PCA-analysis and CIs are plotted.

Details

It compares if the 95% confidence interval of the eigenvalues overlap with the mean for each eigenvalues of a Null model (permuted colums). Only Eigenvalues significantly different than those from the NULL model are retained.

Value

A `list` with the following elements:

- eigen.data - Eigenvalues of the original dataset
- eigen.data.bootstrap - Eigenvalues of the bootstrapped dataset
- eigen0 - Eigenvalues of the Null-model
- Nr.PCs - the number of PCs to retain based on comparing the eigenvalues of the Null-model with the bootstrapped eigenvalues.

References

Horn JL (1965): "A rationale and test for the number of factors in factor analysis." Psychometrika 30:179–185

`fieldAnomaly.raster` *Field anomaly for a spatio-temporal field*

Description

Wrapper function for `fieldAnomaly(...)` from the package sinkr to calculate the daily/monthly field anomaly for a raster brick.

Usage

```
fieldAnomaly.raster(rst1, rst2.anomalies = NULL, time, level = "month")
```

Arguments

`rst1`	A raster either a `RasterBrick` from the raster package or `SpatRaster` from terra with each layer corresponding to a time step.
`rst2.anomalies`	An optional second raster, where anomalies already has been calculated on. Used for correcting the first raster with the same mean anomalies e.g. as a historical reference period to base the correction on.
`time`	A vector of the class "Date", "POSIXct", or "POSIXlt" corresponding to each layer in the raster.
`level`	A character string. The temporal resolution over which to calculate the anomalies. Either `julian` / `day` for daily or `month` for monthly anomalies. Will consider "Day of the year" from as.POSIXlt(x)$yday for `julian` and unique format(x,"%m%d") strings for `day`.

Details

The field anomalies are calculated by subtracting the daily/monthly means from each spatial location of a field.

Value

A raster `brick` containing the field anomalies.

References

Taylor, M. (2017). sinkr: Collection of functions with emphasis in multivariate data analysis. R package version 0.6. https://github.com/marchtaylor/sinkr

Examples

```
# load AHOI SST data
library(raster)
data(sst.ahoi)
# calculate anomalies
sst.anom = fieldAnomaly.raster(sst.ahoi,
                         time = as.Date(sub("X","",names(sst.ahoi)),format = "%Y.%m.%d"),
                               level = "month")
# plot for comparison
par(mfrow = c(1,2))
plot(sst.anom[[1]],las = 1)
```

```
title("SST-anomaly",adj = 0)
plot(sst.ahoi[[1]],las = 1)
title("SST-raw",adj = 0)
```

`fieldAnomaly.to.original` *Add back field anomaly to a spatio-temporal field*

Description

The counterpart of the `fieldAnomaly.raster(...)` function to add back the anomalies and get back the original field.

Usage

```
fieldAnomaly.to.original(rst1, time, level = "month")
```

Arguments

`rst1`	A raster either a `RasterBrick` from the raster package or `SpatRaster` from terra with each layer corresponding to a time step.
`time`	A vector of the class "Date", "POSIXct", or "POSIXlt" corresponding to each layer in the raster.
`level`	A character string. The temporal resolution over which to calculate the anomalies. Either `julian` / `day` for daily or `month` for monthly anomalies. Will consider "Day of the year" from as.POSIXlt(x)$yday for `julian` and unique format(x,"%m%d") strings for `day`.

Details

The field anomalies are calculated by subtracting the daily/monthly means from each spatial location of a field. Therefore in order to get back to the original field the mean on each location is added back to the anomalies.

Value

A raster either of form `RasterBrick` or `SpatRaster` containing the original field.

References

Taylor, M. (2017). sinkr: Collection of functions with emphasis in multivariate data analysis. R package version 0.6. https://github.com/marchtaylor/sinkr

Examples

```
library(raster)
data("sst.ahoi")

# calculate anomalies
time.ahoi = as.Date(sub("X","",names(sst.ahoi)),format = "%Y.%m.%d")
sst.anom = fieldAnomaly.raster(sst.ahoi,
                               time = time.ahoi,
```

```
                                level = "month")
# back calculate the original field
original.field = fieldAnomaly.to.original(sst.anom,time = time.ahoi,level = "month")

# check if they are the same
all(cellStats(original.field,mean) -cellStats(sst.ahoi,mean)== 0)
# correct
```

`fitnessfuns` *Random Forest & extreme randomized Trees multi-obj. fitness functions for use with NSGA-II feature selection*

Description

Various multi-obj. fitness functions for use with NSGA-II feature selection

Usage

```
MO.fitness.func.RF.ranger(
  feat,
  x,
  y,
  seed,
  metric.func,
  mtry = NULL,
  nTree = 200,
  folds = 3,
  reps = 20,
  only.obs.vs.pred = FALSE,
  return.scaled.metric = TRUE
)

MO.fitness.func.extraTrees.ranger(
  feat,
  x,
  y,
  seed,
  metric.func,
  mtry = NULL,
  nTree = 200,
  folds = 3,
  reps = 20,
  only.obs.vs.pred = FALSE,
  return.scaled.metric = TRUE
)
```

Arguments

`feat`	The binary coded feature vector that denotes which gene (feature) is on (1) or off (0).
`x`	The dataframe containing all the features to run feature selection on.

y	Numeric vector of the response variable.
seed	The setting for the seed.
metric.func	Slot for the metric function to be included. The format is metric.func(data) see metricfuns for details.
mtry	Number of variables to possibly split at in each node. Default is the square root of the Number of Variables.
nTree	Number of Trees to build in Forest for Random Forest/ Extreme Randomized Trees algorithm.
folds	Number of folds (k) for n-repeated k-fold crossvalidation.
reps	Number of repetitions (n) for n-repeated k-fold crossvalidation.
only.obs.vs.pred	A logical value. Setting to return only observations vs. predictions without any metric-function being considered.
return.scaled.metric	A logical value. Setting if the metric should be scaled by a naive forecast (mean of all observations).

Details

MO.fitness.func.RF.ranger - Fitness function for the "randomForest" - algorithm from the ranger package.
MO.fitness.func.extraTrees.ranger - Fitness function for the "extraTrees" - algorithm from the ranger package.

Note

fitnessfuns is a generic name for the functions documented.

---

flatten.list *Function to flatten a deeply nested list to a shallow one*

Description

Function to flatten a deeply nested list to a shallow one.

Usage

```
flatten.list(x, verbose = TRUE)
```

Arguments

x	A deeply nested list.
verbose	Toggle printing of different layer nestings to console on or off if function steps through them.

Details

A function that flattens a deeply nested list and returns a shallow flattened one with only one level of nesting. Be aware that the single arguments are returned somewhat out of order, since the degree of nesting can vary between elements.

Value

A shallow `list`

Examples

```
# For an artificial toy dataset
lst = list(a = list(a = 1:10,b = list(1,2,3),c = list(a = list(1:100,NA,list(rep(NA,100))))),
           b = list(data.frame(id = 1:4,val = c(2,NA,3,5)),
                    list(letters,data.frame(m = month.abb,no = 1:length(month.abb)))))
lst_unnested = flatten.list(lst)
print(lst_unnested)
```

`metricfuns` *Various metric functions for performance evaluation*

Description

Various metric functions to assess the fit of a statistical model in a regression context.

Usage

```
RRSE(data)

RMSPE(data)

RMSE(data)

MAE(data)

PseudoR2(data)
```

Arguments

`data`	A `dataframe` containing a column with numeric observations (`obs`) and the prediction (`pred`).

Details

`RMSE` calculates the "Root mean squared error". See Wikipedia for details.
`MAE` calculates the "Mean absolute error". See Wikipedia for details.
`PseudoR2` calculates a "R2"-like metric as cor(obs,pred)^2.
`RMSPE` calculates the "Root mean squared percentage error". See Stackexchange for details.
`RRSE` calculates the "Root relative squared error", which is the RMSE relative to the RMSE of a naive prediction by using the deviations to the mean.

Note

`metricfuns` is a generic name for the functions documented.

Model.metrics.regression_NSGAII

Calculates Model metrics (RMSE,MAE,Pseudo-R2) for output of feature selection NSGA-II

Description

Function to calculate Model metrics (RMSE,MAE,Pseudo-R2) for output of feature selection NSGA-II

Usage

```
Model.metrics.regression_NSGAII(NSGA.II.obj, fitness.func)
```

Arguments

NSGA.II.obj	The output object from the NSGA-II feature selection.
fitness.func	A function used as fitness function in the feature selection process.

Value

A data.frame specifying the solutions within the pareto front, with the following columns:

- ID - the index in 'NSGA.II.obj$pareto.varnames' to which each solution corresponds to
- metric - the metric used to evaluate the fitness of the solution.
- params - a number between 0 and 1 representing the scaled number of parameters used for feature selection
- nr.of.params - An integer. Specifies the number of covariates used in the model.
- dist.to.CoordOrigin - distance of the solution within the pareto front to the coordinate origin (0,0)
- RMSE - the RMSE (with the same level of Cross validation, which was used during the feature selection)
- PseudoR2 - the pseudo R2 (with the same level of Cross validation, which was used during the feature selection)
- MAE - the MAE (with the same level of Cross validation, which was used during the feature selection)

mult.merge

Merge multiple tables

Description

Function to merge various data.frames/list of data.frames to one.

Usage

```
mult.merge(fileA, listFileB, by, by.firstCol = FALSE)
```

Arguments

fileA	A string denoting the name of a data.frame
listFileB	A character vector denoting data.frames or lists containg data.frames, which are merged with fileA.
by	Variable name by which to merge.
by.firstCol	A logical value. If merging should be done by the first column in each data.frame regardless of the name.

Value

A merged data.frame.

NSGA.II_wrapper *NSGA-II Wrapper function to perform multi-objective (2) feature selection*

Description

Performs Bi-objective Feature selection via a specified fitness function

Usage

```
NSGA.II_wrapper(
  x,
  y,
  seed,
  ga.input,
  memoisation = F,
  mutation.rate = 0.1,
  crossover.rate = 0.8,
  remove.overlap = T,
  pop.size = 50,
  offspring.size = NULL,
  max.iter = 150,
  initialize.equal = T,
  init.limit = NULL,
  stop.criterion = NULL,
  n.cores = NULL,
  allowZeroGenes = F,
  ref.point = c(1, 1)
)
```

Arguments

x	A data.frame containing all features that should be tried in the feature selection process.
y	The response variable.
seed	Defines the seed used for the internal loop to make sure to get the same fit with the same features used.

ga.input	A named list list(fitness.func, metric.func) containing the fitness function & the metric used to evaluate the fit (e.g. RMSE).
memoisation	A logical value. If memoisation should be used (TRUE) or not (FALSE). If memoisation is used, function calls with the same input parameters (aka same features) are stored in a Cache and returned if solution is visited again.
mutation.rate	Probability of a mutation (bitflip) within the feature vector.
crossover.rate	Fraction of crossover between parent solutions.
remove.overlap	A logical value. If overlapping solutions should be removed (see Nojima et al. 2005)?
pop.size	Parent population size for the genetic algorithm.
offspring.size	Offspring population size for the genetic algorithm.
max.iter	Maximum number of iterations the algorithm should run.
initialize.equal	A logical value. If initial population should be drawn at random (FALSE) or if it should contain individuals drawn uniformly along the number of features dimension (TRUE).
init.limit	A number. Maximum number of genes to activate in the initial population. This is rather useful if the model fitting is slow for models with lots of features (so you want to limit it for computational purposes).
stop.criterion	Maximum number of iterations without improvement of the solution.
n.cores	Number of CPU-cores to parallelize the external loop (parallelize number of individuals in the population). If NULL the function will try all possible number of cores and chooses the fastest (the evaluation will however take some time, based on the machine).
allowZeroGenes	A logical value. If individuals containing only zeros (no features) should be included in the search (TRUE) or not (FALSE).
ref.point	Vector of length 2, defining the Reference point for the Hypervolume Indicator.

Details

This function implements the non-dominated sorting genetic algorithm (NSGA-II) feature selection for the bi-objective case. It simultanously minimizes the Nr. of features and a measure of performance (e.g. RMSE) for a machine learning model (e.g. Random Forest).

Value

A list with the following elements:

- generation.fitness - data.frame of tracked fitness per iteration
- generation.populations - A list of all individuals within the population per iteration
- generation.offspring - A list of all offspring individuals per iteration
- pareto.individuals - individuals of the pareto-optimal solution
- pareto.varnames - Names of features included in the pareto-optimal solution
- pareto.solution - Summary table of the final pareto-solution
- algorithm.params - Input parameters and other summarized in a list to make output more reproducible
- fitted.dataset - list of x and y used for model fitting

Examples

```
## Not run:
 # This short example takes some time (~ 10min on 10 cores of AMD Ryzen 3700X).
 # To make sure the algorithm converged, increase 'max.iter' and 'stop.criterion'
 # as well increase 'reps' in the fitness function for robustness.
set.seed(12345)
N = 100

# create 4 predictors
AR1  = arima.sim(model = list(order = c(1,0,0),ar = 0.3),n = N)
ARMA1.1 = arima.sim(model = list(order = c(1,0,1),ar = 0.3,ma  = -0.5),n = N)
sine = sin(2*pi/(365)*1:N)
cosine = 0.5*cos(2*pi/(0.2*365)*1:N)

# combine to form the response
y = (1/(1+exp(-0.2*((1:N)-N/2)))-0.5) *0.6*AR1 +
  +c((1:N-N/3)^2)/max(c((1:N-N/3)^2)) * 1*ARMA1.1 +
  1:N/max(1:N) * 2*0.3 * sine * cosine +
  rnorm(n = 100,mean = 0,sd = 0.4*min(sd(sine),sd(cosine)))

# plot
layout(mat = matrix(c(rep(1,8),2,2,3,3,4,4,5,5),ncol = 4,nrow = 4,byrow = TRUE),
       height = c(0.2,rep(0.3,4)))
par(mar = c(4,4,3,3))
plot(y,type = "l",main = "Response")
plot(AR1,type = "l",main = "AR1",xlab = "")
plot(ARMA1.1, type = "l",main = "ARMA1.1",xlab = "")
plot(sine,type = "l",main = "Sine",xlab = "")
plot(cosine,type = "l",main = "Cosine",xlab = "")

predictors = cbind(AR1,ARMA1.1,sine,cosine)

# create several correlated nonsense features with correlations ranging between 0.5 - 0.7
# taken from "https://stats.stackexchange.com/questions/15011/
# generate-a-random-variable-with-a-defined-correlation-to-an-existing-variables"
cors = runif(n = 100,min = 0.5,max = 0.7)*sample(c(1,-1),size = 100,replace = TRUE)
nonsenseVars = matrix(NA,ncol = 100,nrow = nrow(predictors))
for(i in seq(cors)){
  rho   <- cors[i]               # desired correlation = cos(angle)
  theta <- acos(rho)             # corresponding angle
  x1    <- predictors[,sample(1:4,1,TRUE)]        # fixed given data
  x2    <- rnorm(N, 0, sd(x1))      # new random data
  X     <- cbind(x1, x2)         # matrix
  Xctr  <- scale(X, center=TRUE, scale=FALSE)   # centered columns (mean 0)

  Id   <- diag(N)                              # identity matrix
  Q    <- qr.Q(qr(Xctr[ , 1, drop=FALSE]))      # QR-decomposition, just matrix Q
  P    <- tcrossprod(Q)          # = Q Q'       # projection onto space defined by x1
  x2o  <- (Id-P) %*% Xctr[ , 2]                 # x2ctr made orthogonal to x1ctr
  Xc2  <- cbind(Xctr[ , 1], x2o)                # bind to matrix
  Y    <- Xc2 %*% diag(1/sqrt(colSums(Xc2^2)))  # scale columns to length 1

  nonsenseVars[,i] = Y[ , 2] + (1 / tan(theta)) * Y[ , 1]     # final new vector

}
```

```
# input data
x = data.frame(predictors,nonsenseVars)

fitness.func = MO.fitness.func.extraTrees.ranger
# run with RMSE-metric
metric.func.RMSE = marmalaid::RMSE
# define nr. of folds and permutations
formals(fitness.func)$folds <- 5
formals(fitness.func)$reps <- 10
formals(fitness.func)$return.scaled.metric = TRUE

ga.input.RMSE = list(fitness.func = fitness.func,
                     metric.func = metric.func.RMSE)

# run NSGA-II feature selection with extraTrees model
NSGAII.feature.sel = NSGA.II_wrapper(
  x,y,
  seed = 12345,
  ga.input = ga.input.RMSE,
  mutation.rate = 0.1,
  crossover.rate = 0.8,
  remove.overlap = TRUE,
  pop.size = 100,
  offspring.size = 100,
  max.iter = 40,
  initialize.equal = TRUE,
  stop.criterion = 10,
  n.cores = 10,
  ref.point = c(1,1)
)

# plot evolution
plot_NSGAII(NSGAII.feature.sel)
# chosen variables within pareto front
NSGAII.feature.sel$pareto.varnames
# model performance
Model.metrics.regression_NSGAII(NSGAII.feature.sel,MO.fitness.func.extraTrees.ranger)

## End(Not run)
```

---

plot_NSGAII *plot function for output of* NSGA.II_wrapper

Description

A summary plot for the NSGA-II feature selection.

Usage

```
plot_NSGAII(output.NSGAII)
```

Arguments

output.NSGAII	An obj. (list) containing results of the NSGA-II feature selection.

Value

A plot showing the evolution with the Hyper-volume Indicator as well as the Pareto-Front.

`salt.ahoi` *Salinity data of the North Sea*

Description

A raster dataset containing the averaged salinity over the first 10m in the North Sea for the years 1990 - 2014.

Usage

```
data(salt.ahoi)
```

Format

A raster brick:

dimensions latitude, longitude, number cells, time

resolution spatial resolution of the data set

extent spatial extension of the data set

crs Coordinate reference system

names Names of each layer in the time dimension - contains the time step, here denoted with XYYYY.MM.DD

Details

This dataset contains the averaged salinity over the first 10m in the North Sea between -5.1 to 10.1 longitude and 50.5 to 62.1 latitude. It is based on an extraction of the AHOI dataset (v19.02), which spans temperature & salinity data from 1948 - 2014 at 54 vertical depth layers at a resolution of 0.2°x 0.2°.

Source

Thuenen Institute for Sea fisheries, "AHOI : A physical-statistical model of hydrography for fishery and ecology studies" https://www.thuenen.de/en/sf/projects/ahoi-a-physical-statistical-model-of-hydr

References

Núñez-Riboni I, Akimova A (2015) "Monthly maps of optimally interpolated in situ hydrography in the North Sea from 1948 to 2013." J Mar Syst 151:15–34

SOM.DVIndex *Dynamic Validity Index (DVI) of Shen et al. (2005) for SOMs*

Description

Function to calculate the DVI for SOMs with various grid-sizes.

Usage

```
SOM.DVIndex(list.SOM.obj, c.param = 1, control.for.empty.nodes = FALSE)
```

Arguments

list.SOM.obj	A list of SOM objects of class(kohonen) with different map sizes to be evaluated with the DVI.
c.param	Control parameter to weight the influence of within cluster density and in-between cluster seperability.
control.for.empty.nodes	Logical value, denoting if empty clusters should be discarded in the calculation of the in-between distance (TRUE) or not (FALSE).

Details

This function implements the DVI of Shen et al. (2005) for SOMs, which was originally proposed for kmeans clustering. It identifies the optimal number of clusters to be a compromise between intra compactness of clusters and inter-seperatedness. Clusters should have a small distance to members of their own cluster (intra-ratio) as well as having a larger distance to members of other clusters (inter-ratio). The DVI is now the sum of both the normalised intra- and inter ratios (optional scaling with a correction parameter is possible) and has a minimum at the prefered number of clusters. Since larger map sizes of SOMs can lead to empty clusters (nodes that have no datapoint attached to them), a correction with the option control.for.empty.nodes = TRUE is possible, that only considers clusters with data attached to them.

Value

A data.frame with the following columns:

- xdim - the y-dimension of the SOM mapping.
- ydim - the x-dimension of the SOM mapping
- size - the total number of clusters (product of xdim and ydim)
- intra.ratio - A measure of the inner cluster density.
- inter.ratio - A measure of the seperability between clusters.
- DVI - The dynamic validity index, being a combination of intra - and inter ratio.

References

Shen J, Chang SI, Lee ES, Deng Y, Brown SJ (2005) "Determination of cluster number in clustering microarray data." Appl Math Comput 169:1172–1185

Examples

```
library(raster)
data(sst.ahoi)
sst.anom = fieldAnomaly.raster(rst1 = sst.ahoi,
                        time = as.Date(sub("X","",names(sst.ahoi)),format = "%Y.%m.%d"),
                             level = "month")
# calc. mean over summer season
sst.JJA = calc.mean.over.Month(sst.anom,
                             time = as.Date(sub("X","",names(sst.anom)),
                                            format = "%Y.%m.%d"),
                             month = c(6,7,8),shiftYear = FALSE)

# grid sizes
grid.size = data.frame(expand.grid(1:5,1:5))
names(grid.size) = c("xdim","ydim")
SOMs.store = list()
for(i in 1:nrow(grid.size)){
 cat(i,"\n")
 # calculate spatial SOM
 SOMs.store[[i]] = spatial.SOM(x = sst.JJA,time = as.numeric(sub("X","",names(sst.JJA))),
                              plot = FALSE,seed = NULL,
                              parallel = c(parallel = FALSE,cores = NA),
                              reps = 1,return.all = FALSE,
                              grid = kohonen::somgrid(xdim = grid.size$xdim[i],
                                                      ydim = grid.size$ydim[i],
                                                      topo = "rectangular",
                                                      neighbourhood.fct = "bubble",
                                                      toroidal = FALSE),
                              mode = "online",rlen = 1000)
}
# calculate DVI
DVI.SST.JJA = SOM.DVIndex(list.SOM.obj = lapply(SOMs.store,function(x) x$SOM.out),
                         c.param = 1,
                         control.for.empty.nodes = TRUE)
DVI.SST.JJA = DVI.SST.JJA[order(DVI.SST.JJA$size),]
# plot DVI
par(mfrow = c(1,1))
plot(1:nrow(DVI.SST.JJA),DVI.SST.JJA$DVI,type = "l",xaxt = "n",
     xlab = " grid size",ylab  = "DVI",las = 1)
points(1:nrow(DVI.SST.JJA),DVI.SST.JJA$DVI)
abline(v = which(DVI.SST.JJA$DVI == min(DVI.SST.JJA$DVI,na.rm = TRUE)),
       col = "red",lty = 2)
axis(side = 1,at = 1:nrow(DVI.SST.JJA),labels = DVI.SST.JJA$size)
title("DVI",adj = 0)
```

`SOM.qualityMetrics` *Quality metrics for the map representation of a SOM*

Description

Various quality metrics to assess the quality of the SOM-mapping returned by `som`.

Usage

```
SOM.quant.error(SOM.obj)

SOM.topo.error(SOM.obj)
```

Arguments

SOM.obj	An object of class(kohonen) produced via som.

Details

SOM.quant.error - Calculates the Quantisation error (QE) for the SOM mapping
SOM.topo.error - Calculates the Topographic error (TE) (Kiviluoto 1996) and a combined index of QE & TE based on the work of Kaski & Lagus 1996

Value

SOM.quant.error:

A numeric value for the QE

SOM.topo.error:

A named list with following elements:

- TE - a numeric value for the Topographic error
- C.index - a numeric value for the combined metric of QE & TE error from Kaski & Lagus 1996
- Mapping.of.data - A data.frame showing the best matching unit (BMU), the second best matching unit (SNDMU), the C.index and an integer value denoting if BMU and SNDMU are neighbours (1) or not (0) for every data point.

Note

SOM.qualityMetrics is a generic name for the functions documented.

References

Kiviluoto K (1996) "Topology preservation in self-organizing maps." In: Proceedings of International Conference on Neural Networks (ICNN'96). IEEE, p 294–299

Kaski S, Lagus K (1996) "Comparing self-organizing maps." In: von der Malsburg C , von Seelen W , Vorbrüggen JC, Sendhoff B (eds) Artificial Neural Networks — ICANN 96. ICANN 1996. Lecture Notes in Computer Science, vol 1112. Springer, Berlin, Heidelberg.p 809–814

Pötzlbauer G (2004) "Survey and comparison of quality measures for self-organising maps." In: Proc. 5th Workshop on Data Analysis (WDA 2004).p 67–82

Examples

```
data(sst.ahoi)
# calculate field anomaly
sst.anom = fieldAnomaly.raster(rst1 = sst.ahoi,
                         time = as.Date(sub("X","",names(sst.ahoi)),format = "%Y.%m.%d"),
                            level = "month")
# calc. mean over summer season
```

```
sst.JJA = calc.mean.over.Month(sst.anom,
                               time = as.Date(sub("X","",names(sst.anom)),
                                              format = "%Y.%m.%d"),
                               month = c(6,7,8),shiftYear = FALSE)
# calculate spatial SOM
SST.SOM = spatial.SOM(x = sst.JJA,time = as.numeric(sub("X","",names(sst.JJA))),
                      plot = FALSE,seed = 1234,
                      parallel = c(parallel = TRUE,cores = 2),
                      reps = 50,return.all = FALSE,
                      grid = kohonen::somgrid(xdim = 2,ydim = 3,
                                              topo = "rectangular",
                                              neighbourhood.fct = "bubble",
                                              toroidal = FALSE),
                      mode = "online",rlen = 1000)
# calculate QE
SOM.quant.error(SST.SOM$SOM.out)
# calculate TE & C-index
SOM.topo.error(SST.SOM$SOM.out)
```

`spatial.PCA` *EOF analysis (S-mode PCA) for a spatio-temporal field*

Description

Wrapper function for `prcomp(...)` on a spatio-temporal field organised as raster

Usage

```
spatial.PCA(
  x,
  center = TRUE,
  scale. = FALSE,
  spatial.extent = "North Sea",
  plot = FALSE,
  var.threshold = 0.95
)
```

Arguments

`x`	Either a raster `brick` for scalar valued spatio-temporal fields (e.g. SST) with each layer corresponding to a time step or a two-element `list` with each element being a `raster` for vector valued fields (e.g. u & v of Currents). Raster files from both the 'raster'-package (`RasterBrick`) and the 'terra' (`SpatRaster`) will work here. If layer names contain a reference to a date it will be used for writing it out.
`center`	A logical value indicating whether the variables should be centered around the mean.
`scale.`	A logical value indicating whether to scale each gridcell by its variance, if `FALSE` the PCA is calculated on the covariance matrix, otherwise (`TRUE`) on the correlation matrix.

`spatial.extent` Region to which the spatial field should be cropped, either `NULL` for no cropping, an object of class `extent` or the character string `"North Sea"` to cut it to the extent(c(xmin = -7.01,xmax = 12.9,ymin = 48.9,ymax = 61)) corresponding to the North Sea region.

`plot` A logical value if the spatial EOF fields should be returned in a plot after calculation.

`var.threshold` A value, indicating up to which percentage of explained variance the spatial fields should be returned (default = 0.95 (aka 95% variance explained)).

Value

A `list` with the following elements:

- raster - A `raster` for scalar valued input fields or a `list` of `rasters` for vector valued fields. Contains the spatial-EOF fields.
- sp.PCA - A list returned by prcomp containing the output of the PCA with
  `x` - containing the PC timeseries and
  `rotation` - the eigenvectors (aka EOF-weights per grid-point).
- time - the time associated to the PCs

Examples

```
# load AHOI SST data
library(raster)
library(terra)
data(sst.ahoi)
# perform EOF-analysis
SST.EOF = spatial.PCA(sst.ahoi,
                      center = TRUE,scale. = FALSE,
                      spatial.extent = NULL,plot = FALSE)
# plot first EOF with PC timeseries
layout(matrix(rep(c(1,2),each = 2),nrow = 2,ncol = 2,byrow = TRUE),heights = c(0.65,0.35))
par(mar = c(4,4,3,3))
image(SST.EOF$raster[[1]],las = 1,xlab = "lon",ylab = "lat")
maps::map(add = TRUE,fill = TRUE,col = "gray80");box()
title("EOF1",adj = 0)
plot(SST.EOF$time,SST.EOF$sp.PCA$x[,1],type = "l",las = 1,xlab = "time",ylab = "")
title("PC1",adj = 0)

# use the terra package-integration instead
SST.EOF = spatial.PCA(rast(sst.ahoi),
                      center = TRUE,scale. = FALSE,
                      spatial.extent = NULL,plot = FALSE)
# plot first EOF with PC timeseries
layout(matrix(rep(c(1,2),each = 2),nrow = 2,ncol = 2,byrow = TRUE),heights = c(0.65,0.35))
par(mar = c(4,4,3,3))
image(SST.EOF$raster[[1]],las = 1,xlab = "lon",ylab = "lat")
maps::map(add = TRUE,fill = TRUE,col = "gray80");box()
title("EOF1",adj = 0)
plot(SST.EOF$time,SST.EOF$sp.PCA$x[,1],type = "l",las = 1,xlab = "time",ylab = "")
title("PC1",adj = 0)
```

spatial.SOM *Self Organising Map (SOM) for a spatio-temporal field*

Description

Wrapper function for som to perform a S-mode SOM on a spatio-temporal field organised as raster

Usage

```
spatial.SOM(
  x,
  time,
  plot = TRUE,
  seed = NULL,
  parallel = list(parallel = FALSE, cores = NULL),
  reps = 100,
  return.all = FALSE,
  ...
)
```

Arguments

x	Either an object of class RasterBrick from the 'raster'-package or a SpatRaster from the 'terra'-package with each layer corresponding to a time step specified in the argument time.
time	the time step associated with each layer of the raster
plot	A logical value if a simple plot with the convergence of the SOM algorithm and the BMU (best matching unit) time series should be plotted.
seed	Seed settings for reproducibility, either numeric or NULL.
parallel	List, containing two elements: parallel - logical, if analysis needs to be run in parallel mode, cores - number of CPU-cores to run analysis on
reps	Number of repetitions to calculate for the SOM
return.all	Logical. If TRUE all SOMs (specified in 'reps' besides the 'best' one) are returned.
...	additional arguments specified in som.

Value

A named list with the following elements:

- SOM.out - The 'best' SOM mapping found in regards to both QE and TE. Object of class(kohonen). See som for details.
- SOM.quality - The quality of the mapping assessed with both QE and TE.
- SOM.raster - raster of the spatial-SOM fields.
- Freq.pattern - the frequency of occurence of the spatial pattern within the BMU time series
- BMU.ts - a data.frame showing the time and the associated BMU
- return.all.SOMs - if return.all.SOMs is TRUE, a list with all stored SOM objects, otherwise NULL.

Examples

```
library(raster)
data(sst.ahoi)
# calculate field anomaly
sst.anom = fieldAnomaly.raster(rst1 = sst.ahoi,
                        time = as.Date(sub("X","",names(sst.ahoi)),format = "%Y.%m.%d"),
                           level = "month")
# calc. mean over summer season
sst.JJA = calc.mean.over.Month(sst.anom,
                             time = as.Date(sub("X","",names(sst.anom)),
                                            format = "%Y.%m.%d"),
                              month = c(6,7,8),shiftYear = FALSE)
# calculate spatial SOM
SST.SOM = spatial.SOM(x = sst.JJA,time = as.numeric(sub("X","",names(sst.JJA))),
                    plot = FALSE,seed = 1234,
                    parallel = c(parallel = TRUE,cores = 2),
                    reps = 50,return.all = FALSE,
                    grid = kohonen::somgrid(xdim = 3,
                                            ydim = 3,
                                            topo = "rectangular",
                                            neighbourhood.fct = "bubble",
                                            toroidal = FALSE),
                    mode = "online",rlen = 1000)
plot(SST.SOM$SOM.raster)
```

---

sst.ahoi *SST data of the North Sea*

Description

A raster dataset containing the averaged temperature over the first 10m in the North Sea for the years 1990 - 2014.

Usage

```
data(sst.ahoi)
```

Format

A raster brick:

dimensions latitude, longitude, number cells, time

resolution spatial resolution of the data set

extent spatial extension of the data set

crs Coordinate reference system

names Names of each layer in the time dimension - contains the time step, here denoted with XYYYY.MM.DD

Details

This dataset contains the averaged temperature over the first 10m (in the following refered to as sea surface temperature aka "SST") in the North Sea between -5.1 to 10.1 longitude and 50.5 to 62.1 latitude. It is based on an extraction of the AHOI dataset (v19.02), which spans temperature & salinity data from 1948 - 2014 at 54 vertical depth layers at a resolution of 0.2°x 0.2°.

Source

Thuenen Institute for Sea fisheries, "AHOI : A physical-statistical model of hydrography for fishery and ecology studies" https://www.thuenen.de/en/sf/projects/ahoi-a-physical-statistical-model-of-hydr

References

Núñez-Riboni I, Akimova A (2015) "Monthly maps of optimally interpolated in situ hydrography in the North Sea from 1948 to 2013." J Mar Syst 151:15–34

Index